Graphics • C++ • C • Pascal • ASM • 386 • Consulting

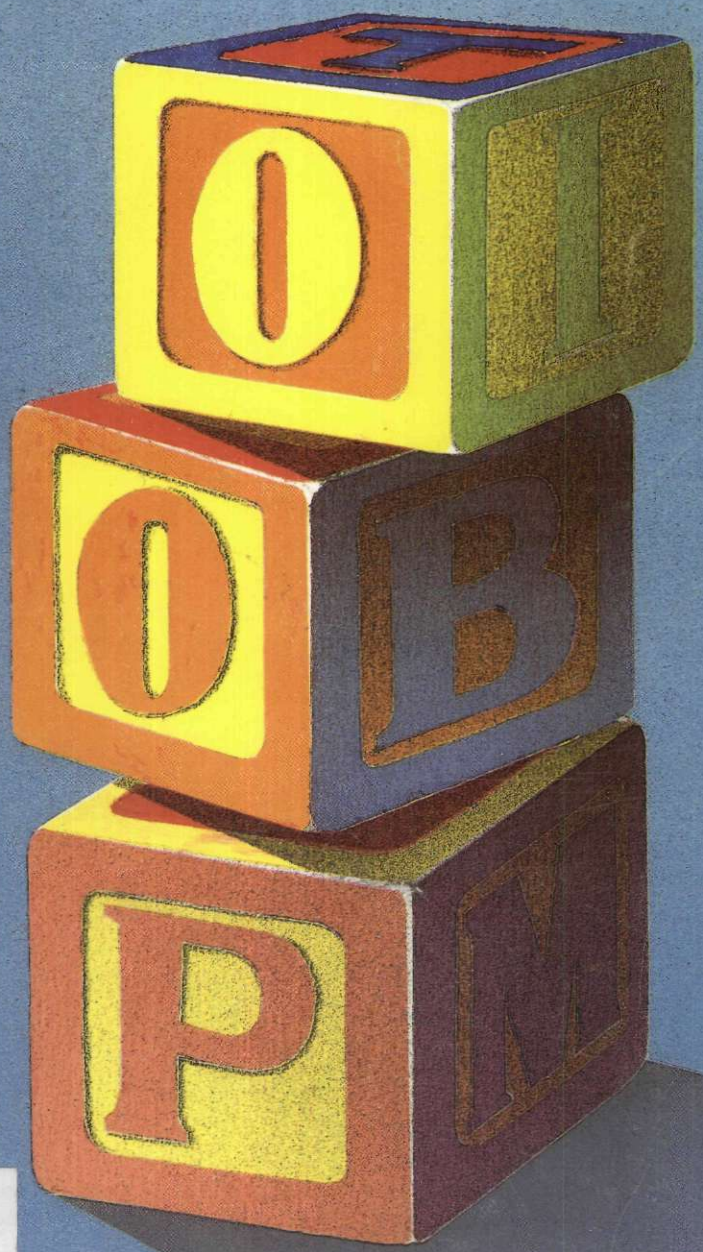
The Resource Journal for IBM Micro Programmers

ASM LIBRARY FEATURE

PROGRAMMER'S Journal™

September/October 1989
Volume 7.5
$4.95 Canada $5.95

OBJECT-ORIENTED PROGRAMMING



CREATING CLASS LIBRARIES IN C++

FROM C TO OBJECTS

BETTER C++ CODE GENERATION

More 256-Color VGA Resolutions

Enhancing the BGI in Turbo Pascal

Zortech and the ANSI C Standard

#21609**** PJS5832 MARAPR90
RON K UNZ
WALL STREET ANALYTICS INC
33-41 85TH ST
JACKSON HEIGHTS NY 11372

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Now it's easy to make your programs millions of times smarter.

More and more, programmers and work station builders are using DESQview as a development tool. The reason is simple. They can create powerful, multitasking solutions today for the millions of DOS PCs in use today. Solutions comparable to those promised for tomorrow by OS/2.

The API Advantage

Programmers who take advantage of DESQview's API (Application Program Interface) get access to the powerful capabilities built into DESQview —multitasking, windowing, intertask communications, mailboxes, shared programs, memory management, mousing, data transfer, menu-building and context sensitive help.

Bells and Whistles

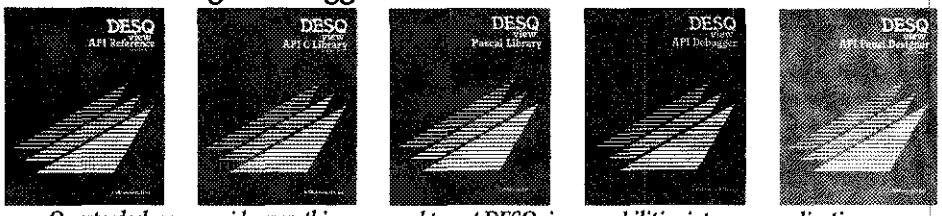
A program taking advantage of the DESQview API can spawn subtasks for performing background operations or new processes for loading and running other programs concurrently. It can schedule processing after an interval or at a certain time. It can use DESQview's intertask communications to rapidly exchange data between programs, share common code and data; or interrupt at critical events. It can use DESQview's menuing and mousing capabilities to create menus. And there's lots more it can do.

Some *of the applications under development right now using DESQview API Tools: CAD, medical systems, insurance, 3270 mainframe communications, network management, real estate, typesetting, point of sale, education, commodity trading, stock trading and online voting.*

80386 Power

Programmers can take advantage of the 80386's protected mode for large programs, yet run on DOS and multitask in DESQview—side by side with other 80386 and DOS programs. The breakthroughs that make this possible: DOS Extenders from PharLap Software and AI Architects and DESQview support of these DOS extenders.

A Complete Interactive Toolkit —Including a Debugger



Quarterdeck can provide everything you need to put DESQview capabilities into your applications.

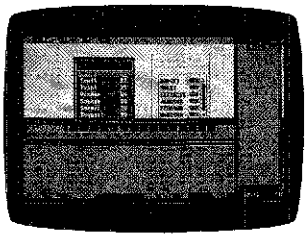
API Reference Manual The key to the power of the DESQview API, this manual contains all you need to know to write Assembly Language programs that take full advantage of DESQview's capabilities. And there's an 'include' file with symbols and macros to aid you in development.

API C Library Here are C language interfaces for the entire set of API functions. It supports the Lattice™ C, Metaware™ C, Microsoft® C, and Turbo C compilers for all memory models. Included with the C Library package is the API Reference Manual and source code for the library.

API Pascal Library The Pascal library provides interfaces for the entire set of API functions. It supports Turbo Pascal V4.0 and V5.0 compilers. Included are the API Reference Manual, source code for the library, and example programs.

API Debugger The DESQview API Debugger is an interactive tool enabling the API programmer to trace and single step through API calls from several concurrently running DESQview-specific programs. Trace information is reported symbolically along with the program counter,registers, and stack at the time of the call. Trace conditions can be specified to report only calls of interest.

API Panel Designer This interactive tool helps you design windows, menus, help screens, error messages, and forms. It includes an editor that lets you construct an image of your panel using simple commands to enter, edit, copy, and move text, as well as draw lines and boxes. You can then define the characteristics of the window that will contain the panel, such as its position, size, and title. Finally, you can specify the locations and types of fields in the panel.



The Panel Designer automatically generates all the DESQview API data streams necessary to display and take input from your panel. These data streams may be grouped into panel libraries and stored on disk or as part of your program.

More Tools are Coming Quarterdeck is committed to adding tools as needed by our users. To that end we have been working with Ashton Tate and Buzzwords International on dBASE III and dBASE IV translators. And in the works, we have BASIC and DOS Extender libraries.

If you'd like hands-on experience with the API Toolkit, write for information on our next API Conference, coming this August.

Bring New Power to DOS with the DESQview API Toolkit.

Quarterdeck

Quarterdeck Office Systems, 150 Pico Blvd., Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

Number 147 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE NEW BENCHMARK IN PROGRAMMING PRODUCTIVITY

The Clarion Professional Developer Can Turn Your Ideas Into Running Applications While Others Are Still Prototyping.

Whether you are new to programming or have a traditional programming background, you'll be amazed by Clarion's Professional Developer. It's an integrated set of software development tools and a high level language that makes the development of customized applications simpler and faster than any previous development tool.

The language is powerful and rich, evolving from languages like Pascal, C, and BASIC. Unlike those languages, however, it was specifically developed to provide you with high level support for features like pop-up windows, tables, menus, reports, and advanced file management. It gives you complete control over every keystroke, every character and color on the screen, every byte in a disk file, and every location in memory. You can even include routines and complex logic written in C and Assembler.

The Clarion Professional Developer has also been described as a 4GL because of its ability to create applications quickly and simply, with much (if not all) of the code automatically generated for you. And its programmable DBMS power has earned it PC Magazine Editor's Choice.

THE END OF TEDIOUS CODING

Thousands of Clarion users report that they develop applications 2 to 10 times faster than before and without a single compromise. One reason is Designer, the powerful application generator that lets you produce complete programs without coding. In one session with Designer, it's possible to design an entire application, prototype it, create fully commented source code (which you can add to or modify in the Clarion Editor), compile and test it, and create an .EXE program -- even one that is LAN ready. Since code is automatically generated, applications are error-free, reducing development time even more.

ADVERTISEMENT

Both the Designer and the Editor also include WYSIWYG screen and report formatters. You'll never code another screen.

TOTAL AND "INTELLIGENT" INTEGRATION

The Professional Developer includes a full set of 13 integrated, interactive utilities. Each utility can be executed from the main menu, another utility, or the DOS >.

All operate similarly, so that you'll only need to use one general set of commands. Each utility knows what is being worked on, and transition between them is smooth and instantaneous.

You can test and completely debug before linking; use the Translator utility to produce .OBJ modules, then use your favorite linker to produce .EXEs. Our intelligent integration reduces your development time.

DATA MANAGEMENT AND BUILT-IN LAN SUPPORT

The Clarion Professional Developer gives you precise control of file relationships, buffers and caching, transaction processing and encryption. There's no practical limit to keys, fields, records or files. DOS and Clarion files are supported. dBASE III files may be optionally processed, in place, without conversion. DIF, dBASE, and BASIC files can also be imported and exported.

If you need to develop network applications, unlimited network support is included in the base price... no "LAN Packs" to buy. And all the leading networks are supported.

LANGUAGE EXTENSION MODULES (LEMs)

Clarion's optional Language Extension Modules provide supplemental procedures and functions for specific requirements. The new GRAPHICS LEM, for instance, provides over 70 graphics procedures and functions, ranging from low-level drawing commands to bar, pie, line, scatter or polar graphing formats with many statistical functions. You can generate time-series data graphs, as well as display graphic images from a disk file.

Other options include the COMMUNICATIONS LEM, allowing you to control up to 4 asynchronous communication lines from within a Clarion program; the dBASE LEM, the FINANCE LEM, and the Zip Code LEM. You can even develop your own special purpose LEM with Clarion's LEM Maker.

COPY AND DISTRIBUTE YOUR PROGRAMS FREE

With Clarion there are no run-time licenses or fees. And there are no required command lines, copyright notices or special menus visible to the application's end user.

TRY BEFORE YOU BUY!

At $695, we're sure you'll find that the Clarion Professional Developer is the best investment you'll ever make.

However, The Programmer's Shop is offering the opportunity to fully work with the product for a period of 30 days. If you are not completely satisfied, return the product. If you like it, you pay only $459 -- a $236 savings. Offer expires 10/31/89.

FOR FURTHER INFORMATION, CALL:

THE PROGRAMMER'S SHOP
800-421-8006

CLARION SOFTWARE™
150 West Sample Road,
Pompano Beach, Florida 33064
Phone 305 785-4555 Fax 305 946-1650

Clarion Software runs on any IBM PC, PS/2 or true compatible with 512K of memory and a hard disk. Clarion Professional Developer and Clarion Software are trademarks of Clarion Software. ©1989 Clarion Software.

Number 299 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal for IBM Micro Programmers

PROGRAMMER'S Journal™

Volume 7.5
September/October 1989

Contents

4 Notes from the Editor

6 Letters
Tom Juntti, Adrian B. Crum, John Navas II, Martin Althaus, Thomas Leathley, W. T. Sumerlin, Jr., and Ben Myers

16 *PJ* Forum: A Technological Schism *by Hal W. Hardenbergh*
Hal takes a historical look at RISC and CISC as the debate goes on

20 Consulting: Source Code Version Control *by Ken Cunliffe*
A seasoned user describes the benefits of Version Management on large software projects

24 On Graphics: Still-Higher 256-Color VGA Resolutions *by Michael Abrash*
Michael shows how to crank standard VGAs up to 360x480 256-color resolution

39 OOP: The Forest for the Trees *by Vaughn Vernon*
Creating class libraries in C++ that cooperate and coexist

44 OOP: From C to Objects *by Steve Sando*
Bridging the gap between traditional and object-oriented programming with a little class

56 OOP: Better C++ Code Generation *by John M. Dlugosz*
A close look at the code generated by C++ translators and compilers and how it could be improved

62 ASM: A Library System for Assembly *by John Otken*
Our coding wizard describes how to use libraries effectively and gives a detailed look at 80x86 register usage

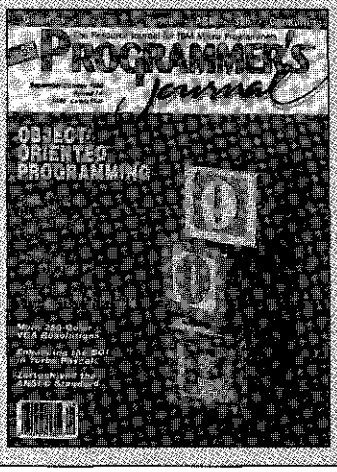
74 Pascal: The World in Your Hands *by Gene Fowler*
Adding world scaling to Turbo Pascal's BGI graphics primitives

82 C: Numeric Extensions, Zortech C++, and ANSI C *by Rex Jaeschke*
Rex discusses numeric extensions to C and takes a close look at Zortech C/C++

87 Miscellaneous *by M. Steven Baker*
386 Development Environments and Tools
Soft-ICE and MagicCVW (Nu-Mega Technologies)
OS/286 and OS/386 Developers' Kits (Eclipse)
FTN77/386 (OTG Systems)
Mini-Case Tools
CC-RIDER (Western Wares)
EasyFlow (HavenTree Software)
Books
Algorithms by Robert Sedgewick (Addison-Wesley)

94 *PJ* Billboard

96 Advertiser Index



No, our cover art doesn't mean that Big Blue made a boo-boo. *OOP* means "object-oriented programming," a relatively new approach to programming that is just beginning to get the attention it deserves. In this issue of *PJ*, some of the best programmers in the field show you how to make child's play of object-oriented programming.

Artist:
Jarrett Jester

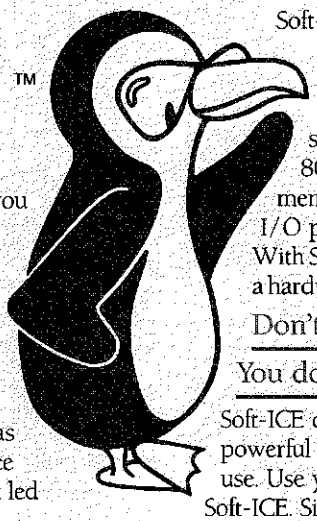
Art Director:
Ralph Bentley

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FINALLY. A debugging tool tough enough to handle the DOS Nasties.

New Version 2.0

Soft-ICE™



Nasty over-write? No sweat!

Soft-ICE memory range break points help you track down memory over-write problems whether you are doing the over-writing or another program is over-writing you.

Hung program? No problem!

When the system hangs, you now have hope. With Soft-ICE you can break out of hung programs no matter how bad the system has been trashed. And with Soft-ICE's back trace ranges you can re-play the instructions that led up to the crash.

Program too large? Not with Soft-ICE!

Soft-ICE runs entirely in extended memory. This means you can debug even the largest DOS programs. And since your program runs at the same address whether Soft-ICE is loaded or not you can find those subtle bugs that change when the starting address of your code changes.

System debugging? Soft-ICE is a natural!

Soft-ICE is ideal for full source level debugging of TSRs, interrupt service routines, self booting programs, DOS loadable device drivers, real-time kernels, non-DOS O/Ss and ROMs. Soft-ICE can even debug within DOS & BIOS.

How Soft-ICE Works

Soft-ICE uses the power of the 80386 to surround your program in a virtual machine. This gives you complete control of the DOS environment, while Soft-ICE runs safely in protected mode. Soft-ICE uses the 80386 to provide real-time break points on memory locations, memory ranges, execution, I/O ports, hardware & software interrupts. With Soft-ICE you get all the speed and power of a hardware-assisted debugger at a software price.

Don't want to switch debuggers?

You don't have to!

Soft-ICE can run stand-alone or it can add its powerful break points to the debugger you already use. Use your favorite debugger until you require Soft-ICE. Simply pop up the Soft-ICE window to set powerful real-time break points. When a break point is reached, your debugger will be activated automatically.

MagicCV with Soft-ICE

Using Soft-ICE with CodeView gives you the features necessary for professional level systems debugging. MagicCV and Soft-ICE can work in concert with CodeView to provide the most powerful debugging platform you will find anywhere.

"These may be the only two products I've seen in the last two or three years that exceeded my wildest expectations for power, compatibility and ease-of-use."

—Paul Mace
Paul Mace Software

Soft-ICE	$386
MagicCV	$199
MagicCV for Windows	$199

Buy Soft-ICE & MagicCV(W) —Save $86.

Buy MagicCV and MagicCVW —Save $100.

Buy All 3 —Save $186.

30 day money-back guarantee
Visa, MasterCard and AmEx accepted

1988 BYTE Award of Distinction

New Soft-ICE 2.0 features

- Back Trace Ranges
- Symbolic & Source level debugging
- EMS 4.0 support with special EMS debugging commands
- Windowed user interface

Nu-Mega™ TECHNOLOGIES

CALL TODAY (603) 888-2386
or FAX (603) 888-2465

RUN CODEVIEW IN 8K MagicCV™

CodeView is a great integrated debugger, but it uses over 200K of conventional memory. MagicCV uses advanced features of the 80386 to load CodeView and symbols in extended memory. This allows MagicCV to run CodeView in less than 8K of conventional memory on your 80386 PC.

NEW—Version 2.0 includes EMS 4.0 driver.
Attention Windows Developers!
Version available for CVW.

P.O. BOX 7607 ■ NASHUA, NH ■ 03060-7607

Number 160 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal for IBM Micro Programmers
PROGRAMMER'S Journal™

NOTES from the Editor

Editor
M. Steven Baker
Managing Editor
Robert E. Wanetick
Contributing Editors
Michael Abrash
Frank D. Greco
Tom Swan

Contributing Writers
Michael Abrash
M. Steven Baker
Ken Cunliffe
John M. Dlugosz
Gene Fowler
Hal W. Hardenbergh
Rex Jaeschke
John Otken
Steve Sando
Vaughn Vernon

Publisher
Elizabeth Oakley
Associate Publisher
Bobbi Sinyard
Production
Ralph Bentley
Scott E. Pringle
Illustration
Jim Carpenter
Advertising
Robert Tappan
Andrea McHone
Subscriptions
Gary Hoyer
Administrative Assistant
Lori Fanslow
Founder
Greg Estes

Business and Editorial Offices
Programmer's Journal
150 North 4th Street
Springfield, OR 97477
503-747-0800
Fax: 503-746-0071

Subscriptions: 1 year (6 issues) $24
($30 in Canada and $40 elsewhere)
Change of Address
Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited for publication.

Code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Robert Tappan or Andrea McHone at 503-747-0800, or write to Programmer's Journal, P. O. Box 30160, Eugene, OR 97403.

Programmer's Journal is produced using the following software products: Ventura Publisher,™ Aldus PC PageMaker,® and Corel Draw. The PostScript™ files are output on a Linotronic™ 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company. Second-class postage paid at Springfield, OR, and additional mailing offices.

POSTMASTER: Send address changes to *Programmer's Journal*, P.O. Box 30160, Eugene, OR 97403.

© 1989 OAKLEY PUBLISHING COMPANY

Object-oriented programming (OOP) and object-oriented design (OOD) are apparently this year's buzzwords. Some folks predict that OOP languages will soon sweep functional programming languages into the past. Others argue that OOP will be the preferred approach for certain specific applications that fit the mold. In any case, lots of reasons are cited for the need to move to OOP languages and OOD concepts.

Information hiding and code reusability are often mentioned. Compared with the functional programming languages used in the past, data and objects (instances of data) become the central unifier. Code and data become commingled in the programming context. But clearly a strong impetus is the increasing complexity of the newer computer environments and systems—OS/2 with Presentation Manager, UNIX with X-Window, News, OSF/Motif, and Open Look graphical interfaces. OS/2 with PM has over 700 system calls to comprehend, a mind boggler for sure.

Most OOP languages may solve these complex problems easier than C or Pascal, but at the machine level, some performance penalty is usually inevitable. On slow CPUs with limited memory (older PCs), this performance reduction might be noticeable. But with the ever-higher CPU speeds and larger memory spaces, it is presumed that wringing out performance is no longer the issue for the user. The programmer's problem has become one of interfacing with such complex environments.

Meanwhile, Back at the Ranch . . .

While OOP may be the buzzword, both large and small firms are still churning out new or updated conventional programming products. Microsoft has recently released an updated DOS FORTRAN compiler and a combined QuickC and QuickAssembler. MetaWare and Lattice have released new versions of their C compilers. Both Borland and Microsoft are dueling it out with Turbo Pascal 5.5 and QuickPascal, both with OOP extensions. Microsoft has also released the Presentation Manager Toolkit, a much less costly version of the OS/2 SDK less the C compiler and assembler.

The DOS OOP market has until recently consisted of implementations of SmallTalk (Digitalk), C_Talk (CNS), and Actor (Whitewater Group). Other options now include Objective-C and Eiffel (a new OOP language from France currently on UNIX/386). An ever-increasing list of vendors—Zortech, Intek, Guidelines, and Glockenspiel (OASYS and Imagesoft)—offer C++ compilers. AT&T missed it scheduled June 30 deadline to release version 2.0 of C++. But clearly the PC C++ battle will be fought next year, when Microsoft and Borland will likely unveil their C++ offerings.

Back at the Video Palace

Before the dust has settled on new BIOS standards for the higher-resolution 800x600 VGA modes, Western Digital has begun sampling their 8514/A chip set. Several other vendors will soon follow. Just as the VGA begins to displace those massive numbers of EGA cards, the new IBM 8514/A-compatible video card will hit the streets. Meanwhile, Texas Instruments has completed the development of TIGA-340, and Microsoft has written fast TIGA drivers for PM and, presumably, Windows 3.0. This fall should be a busy video season.

Should You RISC It?

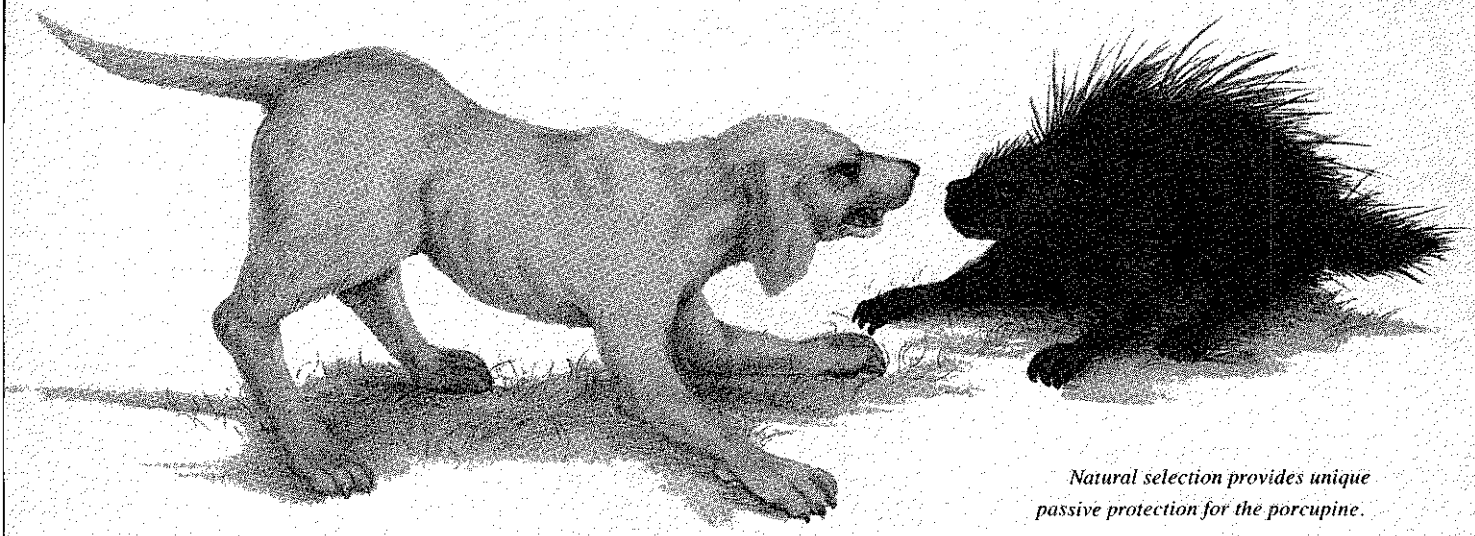
Spring saw the official introduction of the 80486 chip, the Intel i860 chip, and a score of powerful, low-priced RISC and CISC workstations from SUN, DEC, Data General, and others. By fall, we may actually see some of these new workstations rather than just samples. And by 1990, we should likely see machines using the new CPUs. The 80486 chip held few surprises. The math coprocessor moved on board, but no new instructions were added. Although current versions of the chip have some status bits reversed, these will be corrected before the 486 is released in a machine. Intel has clearly targeted 386 compatibility as foremost. This may give software developers a chance to "catch up" with hardware developments.

DOS extenders are becoming more entrenched in the PC marketplace. Both AutoCAD and VersaCAD for the 386 are available. The PC version of Interleaf Publisher has reduced memory needs from 6 to only 2 megabytes of memory using the Phar Lap DOS extender with the Virtual Memory Manager. OS/3 still appears far into the future, while EISA is still off to a slow start until fall COMDEX.

Intel has also been effective (with the 80386) breaking into the UNIX market once dominated by Motorola. Motorola has finally started to sample 50-MHz 68030s this summer (Definicon has a 50-MHz DSI coprocessor board running). But now the workstation vendors are targeting the high-end PC market with their low-cost machines. This should be an interesting show to watch. If you're fortunate enough to be in the memory business, it really matters not who wins. —*Steve* PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Software Developers



Natural selection provides unique passive protection for the porcupine.

The Activator - Natural Selection For Software Protection

Inventor and entrepreneur Dick Erett explains how "The Activator" provides sane protection for your intellectual property.

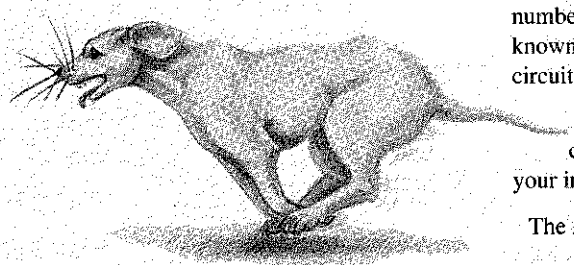
"In any industry, just as in nature, the process of natural selection raises one solution above another. Natural selection is the most elegant of engineers.

In the area of software protection The Block has been selected by the marketplace as the solution that works. Over 500,000 packages are protected by our device.

For the past 4 years our philosophy has been; *'You have the right and obligation to protect your intellectual property.'*

A New Ethic For Software Protection

In allowing end-users unlimited copies of a software package and uninhibited hard disk and LAN operation, The Block has created a new ethic for software protection.



By removing protection from the magnetic media we remove the constraints that have plagued legitimate users.

They simply attach our key to the parallel port and forget it. It is totally transparent, but the software will not run without it.

A New Technology For Software Protection

Our newest model, The Activator, builds on our current patented design, and establishes an unprecedented class of software protection.

We have migrated and enhanced the circuitry of The Block to an ASIC (Application-Specific Integrated Circuit) imbedded in The Activator.

This greatly improves speed and performance, while reducing overall size. Data protection can also be provided.

Programmable Option

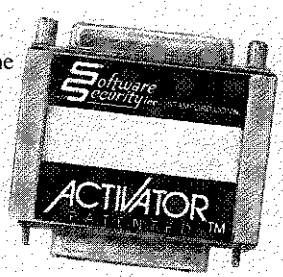
The Activator allows the software **developer** the option to program serial numbers, versions, or other pertinent data known only to the developer, into the circuit, and access it from the program.

Once you program your part of the chip, even we have no way to access your information.

The ASIC makes emulation of the device virtually impossible. It also presents an astronomical number of access combinations.

Full 100% Disclosure

Since The Activator is protected by our patent we fully disclose how it works. Once you understand it, endless methods of protection become evident.



Just as no two snowflakes are the same, no two implementations of The Activator are identical. And like the snowflake the simplicity of The Activator is its greatest beauty.

We never cramp your programming style or ingenuity. Make it as simple or complicated as you desire.

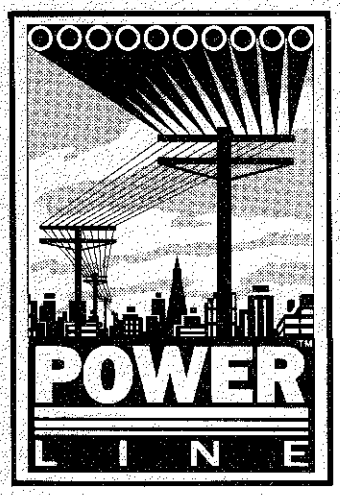
Let us help safeguard what's rightfully yours. Please call today for additional information or a demo unit. *It's only natural to protect your software.*"

1-800-333-0407 ext.103
In Connecticut 203-329-8870
Fax 203-329-7428

Software Security inc.
1011 High Ridge Road
Stamford, CT 06905

Unlimited Copies • Programmable • No Batteries • Small Size • Fast • Patented • Data Protection

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Shove These In Your PC!

Pop POWERLINE II cards into your PC and drive them with our COMMANDO software. Get the results you want from your VOICE PROCESSING APPLICATIONS.

- **Voice Messaging**
- **Call Processing**
- **Audiotex**
- **Telemarketing**
- **Order Processing**
- **Call Distribution**

Our POWERLINE II multiple-port board functions as the heart of a Voice Communications System based on the IBM PC/XT/AT/386 architecture. Each card provides dedicated hardware support for voice processing, call analysis, touch-tone & rotary interaction, and telephone/PBX/Centrex integration.

COMMANDO's extensive function library allows you to rapidly generate sophisticated inbound and/or outbound applications supporting as many as 16 lines—even in the background!

And, all Talking Technology products carry a 2 YEAR WARRANTY. We design them to last!

POWERLINE II $599.00
COMMANDO $139.00
SINGLE LINE from . . . $295.00
VISA/MC/AMEX/COD

FOR ADDITIONAL INFORMATION OR SALES CALL:
(415) 652-9600
FAX (415) 652-5311

TALKING TECHNOLOGY, INC.
4383 PIEDMONT AVE., OAKLAND, CA 94611
Number 297 on the Reader Service Card

On Neural Nets and Graphics

[NOTE: The following is a combination of two letters received from the writer.]

[5/4/89] I have a few comments on Waite and Hardenbergh's "Neural Nets" (*PJ* 7.3). First, the connectionist paradigm is even older than the authors report; the first published papers appeared in the 1940s as biological hypotheses to explain associative memory (e.g., Hebb, D. O., *The Organization of Behavior*, J. Wiley & Sons, New York, 1949).

Second, for a more leisurely introduction to the topic, I recommend Caudill's "Neural Network Primer" series, appearing at irregular intervals in *AI Expert* beginning in December 1987.

Third, the authors' statements, choice of an example network, and estimates of computer horsepower required may convince PC programmers not to explore this interesting field. Caudill's back-propagation network, described in *AI Expert* (June 1988), can be implemented practically on an underpowered 4.77-MHz 8088 machine.

Fourth, the author's definition of a training iteration may influence users to train once on the entire set of exemplars, then twice, etc. I have found it best to train repeatedly with one exemplar until the correct output is produced reliably, then with a second until the net can differentiate the two patterns, then with three, etc. I trained the Caudill example, a 35x4x8 net without biases or momenta, which maps 7x5 pixel bit maps for letters onto their ASCII codes, to nearly its theoretical limit in one afternoon, without automating the training. This example is certainly of real-world significance and offers hope for PC applications of neural nets if, as the authors warn, one can limit the size of the input.

[5/25/89] I'd like to thank Michael Abrash for his fine series, "On Graphics." I've used most of it. In the installment in *PJ* 7.3 is a code listing for an assembly version of Bresenham's line algorithm. I didn't believe that accumulating pixels at the cost of a jump was more efficient than saving the jump but performing more writes to video memory. I tested both versions on a 4.77-MHz NEC V20 machine with a Video Seven VEGA card. The accumulator version required an average of 5.32 seconds to draw 480 pixel lines corresponding to each of the 480 possible angles in the first octant. The jumpless version performed the same task in an average of 5.48 seconds. The difference amounts to 3.3 clocks per pixel in favor of the accumulator version. I also compared video-memory reads to those for ordinary RAM. I found a difference of about 26 clocks per byte read.

I've also always wondered when it is more efficient to shift by one rather than shifting by CL. I guessed the two methods were about even at four shifts. I tested both methods with the queue full and empty at the start of the operation and with three and four bytes to be consumed after the operation. In every case, shifting by CL was superior at three shifts, with execution times changing very little from one to four shifts. When shifting by one with the queue empty at the start, adding an additional shift cost exactly eight clocks. The two clocks required to execute a shift by one are insignificant relative to the eight or more required to fetch the two-byte instruction. A big reason for the surprising performance of shifting by CL is that it allows the queue to fill, three bytes for shifting once and four for shifting twice or more; whereas repeated shifting by one quickly depletes the queue.

As most instructions are queue depleters, as a general rule, the fastest assembly code will be that which executes the fewest bytes of instructions rather than the fewest execution clocks.

Tom Juntti
Whitewood, South Dakota

A Quick Strip Method

Please allow me to share with your readers an assembly language code-optimization trick I discovered recently. While writing a parsing routine, I needed a fast method of stripping spaces from a text string. The following code fragment is one method of accomplishing this:

```
        MOV     CX,STRING_LEN
TOP:    LODSB
        CMP     AL,' '
        JE      NEXT
        STOSB
NEXT:   LOOP    TOP
```

The problem with this code is that two jumps are made whenever a space character is encountered. The code fragment below is the optimization that I came up with:

```
        MOV     CX,STRING_LEN
        JMP     SHORT   TOP2
TOP1:   DEC     CX
        JZ      DONE
TOP2:   LODSB
        CMP     AL,' '
        JE      TOP1
        STOSB
        LOOP    TOP2
DONE:
```

I counted clock cycles from the LODSB instruction through the JE condition to the next LODSB instruction and calculated an eight-percent speed improvement for the second code. Timing tests revealed a range of five to eleven percent in speed improvement, depending on the ratio of space characters to other characters.

Adrian B. Crum
Fontana, California

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Abrash Trapped?

Although I have great respect for Michael Abrash's graphics, assembly programming, and writing skills, in "Fast Line Drawing for the EGA and VGA" (*PJ* Volume 7.3) I fear he fell into the assembly language trap (to use his word) when making his conclusions. His assembler code does indeed outperform his C code by a wide margin. However, he didn't give C a fair shake in comparison to assembler because he failed to implement the algorithm as skillfully in C, and also failed to use Jim Mackraz's suggestion in C.

To demonstrate the difference, I rewrote his EVGALINE.C and incorporated Jim Mackraz's suggestion. *[Interested readers may contact PJ's managing editor for a copy.]* Even though my C code is similar to Michael's assembler code and is more compact than his C code, I submit that I did not sacrifice clarity for speed. To keep the comparison fair, I (mostly) resisted the temptation to make obvious additional improvements. Yet, when compiled with Microsoft C 5.1 and timed on my AST Premium/286 and SOTA VGA/16, my C code is *2.3 times faster* than Michael's C code. (The difference when compiled with Turbo C 2.0 is not so great, owing to less-effective compiler optimization.)

Am I suggesting that C can *always* be just as fast as assembler? Of course not. Michael's assembler code is still over 70 percent faster than my C code, a difference that is not surprising given his great skill. However, less skillful assembler code might well have been slower. In my experience, good C code is easier for average programmers to produce than good assembler code, and the speed of good C code is comparable to the speed of good assembler code in most applications.

In conclusion, I don't agree that assembler is the *only* answer, even for graphics, although I do agree that a good algorithm alone is not a guarantee of good performance. Skillful programming and a good compiler are equally important.

By the way, another excellent reference that is more practical for EGA/VGA graphics programming is the *Programmers's Guide to PC & PS/2 Video Systems*, by Richard Wilton (Microsoft Press, 1987).

John Navas II
Dublin, California

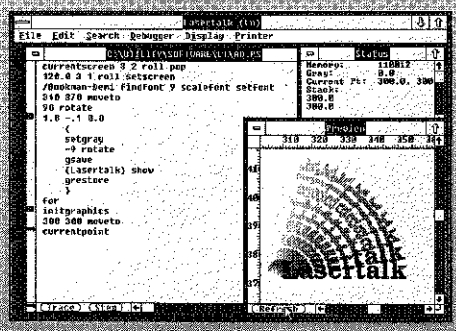
MICHAEL ABRASH RESPONDS: De gustibus non est disputandum. *(For those few non-Latin-speaking programmers out there, that translates as "There is no disputing tastes.") As usual, everything John says is correct—but that doesn't mean I was wrong. To elucidate:*

John is bang-on right that I didn't give the C implementation a fair shake. I explained why in the article: Why spend time optimizing C when optimized assembly language is bound to be much faster? (By the way, while John's C implementation is certainly readable, it does reflect an expert knowledge of coding for speed in C. Honestly, how many programmers would think to OR *the result of* outp *with display memory, in order to avoid an immediate operand? Most programmers would also reflexively count up from 0 to the line length in for loops rather than using the more-efficient counting down.) That brings us to my second point, which is that the 70-percent advantage of assembly over John's fastest C implementation is still a tremendous advantage. My 386 machine at work is just about that much faster than my 286 machine at home—and believe me, I can tell the difference!*

Also, the assembly language line-drawing code itself is probably close to twice as fast as John's fastest C code. The missing 30 percent of that performance advantage is absorbed by a combination of display-memory wait states (about

Continued on page 10

Master PostScript®
Language Programming - On Your PC



Lasertalk™ PC

A PostScript language development environment which provides interactive access to the PostScript interpreter, a PostScript language debugger/editor, on-screen previewer, dictionary browser and Adobe Systems' operator descriptions, online. Requires PC/AT or greater and a PostScript printer. Includes Microsoft® Windows runtime. Suggested Retail Price $399. For more information, see your dealer or contact Emerald City Software at (415) 324-8080.

EMERALD CITY SOFTWARE
800 Menlo Ave, Suite 102, Menlo Park, CA 94025
(415) 324-8080

Lasertalk is a trademark of Emerald City Software. PostScript is a registered trademark of Adobe Systems, Inc.
Microsoft is a registered trademark of Microsoft Corporation.

Number 241 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WHAT'RE YOU GONNA DO?

When your program's full of bugs and you don't know what to do —

Do what the pros do. Go to *Programmer's Journal.*

When professional programmers need help writing better code, they always go to *Programmer's Journal.* They know they'll find the best articles on Ada, ASM, BASIC, C, C++, COBOL, and Pascal written by the industry's top programming talent. Popular authors like Michael Abrash, Steven Baker, Paul Barkley, Ray Duncan, Frank Greco, William Hall, Hal (*DTACK Grounded*) Hardenbergh, Rex Jaeschke, John Newlin, Jake Richter, Richard Riehle, Tom Swan, and Ethan Winer share their technical expertise on languages, operating systems, the 80386, graphics drivers, the VGA, and more! **So if you want to write better code, do what the pros do . . . go to *Programmer's Journal.***

"For hot programming tips look to . . . the excellent . . . *Programmer's Journal.*"
Peter Norton — *Inside the IBM PC*

". . . stuffed with useful information, it definitely deserves a look."
Ray Duncan — *Dr. Dobb's Journal*

"*[Programmer's Journal]* is full of practical information and guidance presented with superb clarity and perspective."
Robert Neff — Computer Press Association

If purchased at the newsstand, the regular price for 6 issues of *PJ* is $29.70, but you can save 33% by starting your one-year, 6-issue subscription today for only $19.95. Do what the pros do . . . go to *Programmer's Journal* and save almost $10.00.

Get a *PJ* Code DiskScription.

What's a *PJ* Code DiskScription? All the source code listings from 6 consecutive issues of *Programmer's Journal*, sent directly to you every other month on disk.

Get all the upcoming installments of Michael Abrash's VGA ASM code, plus *PJ*'s regular offerings of great Ada, ASM, BASIC, C, C++, COBOL, and Pascal source code.

You'll receive your code disks every other month by mail. Learn from *PJ* code or use in it your own programs. Each disk contains all of the program listings published in *PJ* plus .EXE files, .OBJ files, benchmark source files, and more!

If purchased separately, the regular price of 6 code disks is $60.00. Save yourself time, effort, and over $20.00 — order your *PJ* Code DiskScription today for only $39.95!

Guarantee

If at any time you become dissatisfied with your *Programmer's Journal* subscription or your *PJ* Code DiskScription, we will gladly send you a refund covering all unmailed issues or disks.

Offer good in U.S. only. Call or write for prices outside the U.S. Please allow 6 to 8 weeks for delivery.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Free Yourself.

Chains are not only made of iron. There are the shackles of not enough resources. Mismanagement. And the everlasting,"...because that's the way we do things around here."

Break out. Join Microsoft and free yourself. We'll give you all the resources you want. Backed by management that is legendary in microcomputer systems, applications and languages. The chance to work in small, fast-growing groups. Even a *real* office, with a door, so you have the freedom to think.

Apply now. After all, what do you have to lose, except your chains?

Technical Leads

You can be a key member of one our entrepreneurial teams that develop leading-edge software for Microsoft in our OS, Networking, Windows, or Windows/Presentation Manager applications groups.

We're looking for expert designers with technical lead experience to lead a team of experienced developers in the design and development of the product. You should have a BS or MS in CS and preferably 3 or more years of software development experience, with 1 or more years as a successful technical lead. Advanced knowledge of OS internals, networking, windowing environments, or applications design is desired.

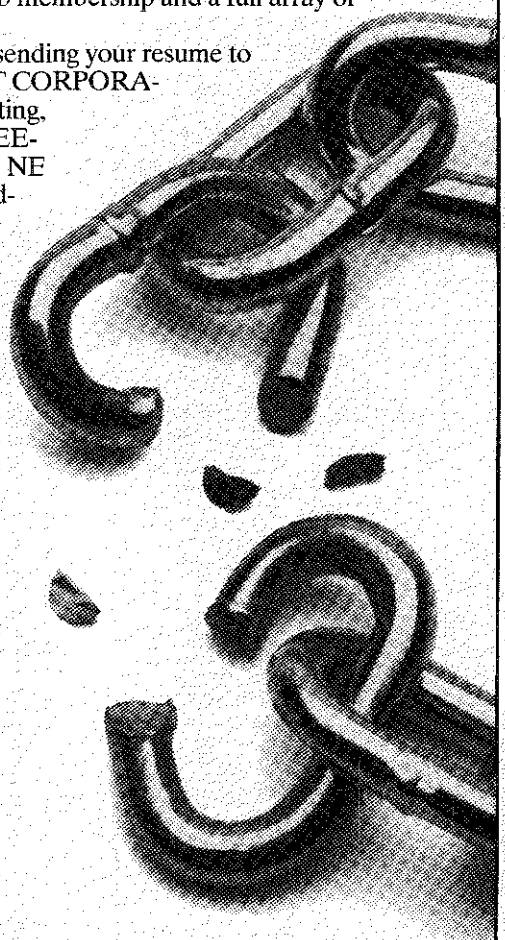
Software Design Engineers

As a developer, you can work on everything from object-oriented methodology, Windows, operating systems, sophisticated graphics, networking, powerful applications software and more. In fact, we're working on some truly visionary ideas we can't even reveal yet.

You could be, too, if you have programming experience and a background that includes micro's, "C" or Pascal, 8086, 68000, UNIX/XENIX™, Windows, Macintosh® Toolbox, or MS-DOS.

Microsoft offers you the ability to live and work where the quality of life is high and the cost of living is low—the beautiful Pacific Northwest. We'll also throw in such amenties as a health club membership and a full array of benefits.

Begin by sending your resume to MICROSOFT CORPORATION, Recruiting, Dept. CWFREE-0989PJ, 16011 NE 36th Way, Redmond, WA 98073-9717. No phone calls, please. We are an equal opportunity employer.



Microsoft®

All Trademarks are registered to thier respective companies.

Number **232** on the Reader Service Card

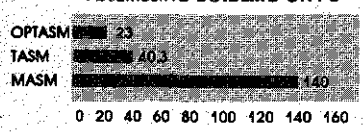
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE PROGRAMMER'S CHOICE

OPTASM
- fastest assembler available
- eliminates phase errors
- never inserts spurious NOPs
- complete TSR online help
- includes a linker and debugger
- "OPTASM is clearly technically superior to MASM."
  Computer Language - Dec 88

$125

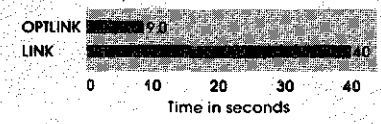
ASSEMBLING **BGIDEMO** ON PC



OPTLINK
- fastest linker available
- generates EXE, COM or SYS
- generates global XREF
- supports CodeView info
- works with MS C5.1

$125

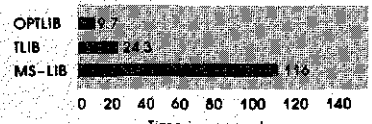
LINKING **BRIEF**



OPTLIB
- 10 times faster than LIB
- * and ? in filenames
- time-based replace
- LIB-compatible syntax
- rename PUBLICs and EXTRNs

$49

BUILD 670K .LIB FILE



OPTDEBUG
- source level - supports CV info
- online help system
- reverse execution
- 50k COM file
- debug EXEs, COMs and TSRs

$125

SLR SYSTEMS
1622 N. Main Street Butler, PA 16001
412-282-0864 TELEX 559215

800-833-3061 (USA)
Call today to order!

Number **165** on the Reader Service Card

three-quarters of a second on an EGA for the sample program), which take about the same amount of time for all versions of the line-drawing code, and by the overhead of the calling C code, which again is the same for all versions.

That's picking nits, though. What's important is that line drawing is the sort of low-level, critical function—basically a system function—that should be done with great care once and then used many times. That 70-percent increase doesn't just make one program faster—it makes every *program that uses line drawing faster, and that covers most graphics programs. Granted, not everyone can write good assembly language; one of the reasons I wrote the article is so that people who want good graphics performance but not the hassle of programming in assembly can speed up their programs. (And, honestly, have you ever known any graphics program that wouldn't benefit from running faster?)*

All that being said, John is still absolutely right that good C code is easier for average programmers to produce. No argument there. Heck, it's easier for anyone *to produce—I do most of my programming in C, and I don't know anyone who uses assembly for the bulk of his or her work. And, yes, good C code* is *comparable to good—not great, but good—assembly code in most applications, and often whatever difference exists between the two isn't worth the extra effort of assembly language.*

So what's to disagree about? Nothing, really—basically, we're just talking about different things. Maybe my mistake was in saying "Truly good PC code requires both a good algorithm and *a good assembler implementation." What I should have said was, "Superb, knock-your-socks-off, dazzle-and-delight-users and reuse-until-the-cows-come-home graphics code requires both a good algorithm* and *a good assembly implementation." If you're writing a one-time-only application, use whatever language gets the job done fastest and most reliably, as long as the code runs adequately well. If you're putting a program with less-than-instantaneous response and screen-update times out there into a competitive segment of the commercial market, or if you need—*not want, *but* need—*speed, use assembly language. If you're not up to writing that code, perhaps you should take up knitting. Or COBOL. Whatever. (That's a generic "you," not intended to mean John Navas, who has shown in past articles and letters that he is an excellent programmer and need not consider knitting.)*

In short, John is right: For most programmers, most of the time, C (or Pascal, or Modula-2—I'm no language fanatic) is a good choice. But I'm also

Would you like to be able to include a hypertext help system in your programs like the big guys: **Microsoft and Borland?***

Introducing... Help System II

We've enhanced the hypertext help system our hypertext products have used for 2 years so that you can include them in your programs:

- **Context Sensitivity**
- **Help for 84 different key combinations**
- **Requires less than 10K**
- **Data compression up to 80 percent**

$199.95
buys object files for Pascal, C, Basic and technical support

To order: **(203) 368 - 0632**
NTERGAID Inc.
955 Connecticut Ave.
Bridgeport, CT 06607

*** At least one of these companies purchased our product before they released theirs.**

Number **302** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE PROGRAMMER'S SHOP

helps save time, money, and cut frustrations. Compare, evaluate, and find products.

NEW DISCOVERIES

Steve Straley's Toolkit Release 2 by Four Seasons Publishing. 543 .LIB routines for Clipper cover windows; menus; pop-up calculator; pop-up help system; arrays; backup and restore; database generation; screen color; table generation, browsing and modifying; trigonometry; financial; PACKing .DBTs; date and time; find files; directory tree; word processor; report writer; and more. Source code debugger, sample programs, reader file for Tom Rettig's Help included. PCDOS **PS price $149**

RR Code. Generate dBASE report programs automatically from R&R Reportwriter reports. Supports multifile relations, free-form page layout, word-wrapped text, complex calculations, totals, subtotals, sorting and grouping, mail merge, complex queries, user-defined functions, print attributes. Documentation optional. **PS price $129**

PageGarden. Uses description files to provide desktop publishing features for ASCII files printed on LIP LaserJet Plus series II/IID or any PostScript laser printer. Use shadings, embedded PCX graphics, sideways printing and more. More than three dozen predefined description files and eight Bitstream fonts included. **PS price $89**

HEADROOM by HELIX Software. Increases apparent amount of memory available by swapping up to 32 applications and TSRs to and from disk, expanded memory or extended memory. Prevents TSR conflicts. Compatible with LIM 3.2 and above, including EEMS. Completely transparent to applications and TSRs, only takes up 50K. Compatible with MDA, Hercules, CGA, EGA, VGA, MCGA and compatible display adapters. **PS price $89**

386 Development Tools	List	Ours
386 Max	75	69
DESQview 386	190	169
FoxBASE +/386	595	399
High C - 386	895	Call
Hoops/386-32	575	489
VM/386	245	219
AI-Expert Systems		
Exsys Professional	795	695
Logic Gem by Sterling Castle	198	179
VP Expert	249	219
AI-Languages		
ARITY Combination Package	1095	979
PC Scheme LISP- by TI	95	79
TURBO PROLOG V2.0	150	115
Turbo Prolog Toolbox	100	75
Assemblers		
MS Macro Asm	150	99
Turbo Assembler/Debugger	150	115
Visible Computer 80286	100	89
BASIC & Addons		
BAS-C Commercial	375	329
dB/Lib	139	125
MS QuickBASIC V4.5	99	69
QBase	149	129
QuickPak Professional	149	129
Softcode	80	70
C Language - Compilers		
High C 286 - by MetaWare	595	559
Instant C/16M	795	719
Lattice C -	450	289
Microsoft C5.1 - w/CodeView	450	299
Micorsoft QuickC	99	69
w/serial mouse	199	99
Turbo C - by Borland	150	Call
CASE & Prototypers		
Dan Bricklin Demo II	195	179
Interactive EasyFlow	150	119
Instant Replay - Nostradamus	150	139
Matrix Layout - flow chart	150	139
MetaDesign by Meta Software	350	329
Pro-C	495	459
Show Partner F/X - demos	350	329
Visable Analyst	595	585
COBOL		
MS COBOL V3.0	900	Call
Realia COBOL	995	849
Communications Addons		
C Asynch Manager - Blaise	175	135
Comm Library by S. Mountain	185	149
Greenleaf Comm Library	229	169
DataBase Management		
CLARION - complete envt.	695	499
dbFast	99	89
D the data language	395	359
Magic PC - visual database	299	269
Paradox V3.0	725	549
R-BASE for DOS	725	539
Dbase		
Clipper Summer '87	695	469
dBASE IV	795	Call
dBMAN V	190	179
FoxBASE + - V2.1	395	249
QuickSilver Diamond	599	379

Dbase Addons	List	Ours
AdComm 99	129	119
Artful Lib	150	139
Clipper Toolbox	295	269
CodeBASE	194	179
dGE - business graphics	195	179
dQuery	150	139
FLIPPER Graphics Library	195	179
Pro Clip	149	125
SilverComm Library	150	139
SilverPaint	100	89
Tom Rettig's Library	100	80

FREE!
Fall catalog.
Over 1,100 products. Revised expanded descriptions make product selection even easier.
MENTION "PJ989"

Dbase Tools		
Buzzwords Toolboxes	295	269
CLEAR + for dBASE	200	169
dAnalyst Plus	195	189
Dbase Online - 6 pop-up refs.	99	89
dBRIEF w/BRIEF	275	Call
dBug - source debugger	195	179
dBX - dBASE III to C	550	529
dSalvage	100	89
Genifer - code generator	395	259
R&R Relational Reportwriter	149	129
Scrimmage by IDL - screen/menu	149	119
Sycero db - single user	495	449
UI Programmer - Dev's Version 2.0	595	479
Debuggers/Disassemblers		
Dis Doc	130	119
Periscope II & III - breakout switch	Varies	Call
SoftProbe II TX - debug	395	359
Sourcer	100	89
Development Tools		
AsmFlow	100	89
C Programmers Toolbox	80	69
Clear Plus for C	150	139
Codan	395	349
Inside	125	119
MKS Lex & Yacc	249	219
MKS RCS	189	169
Poly Doc-SU	199	179
PC-Lint	139	109
Plink 86 Plus - overlays	495	299
PolyMake	149	139
PVCS Corporate	395	369
.RTLINK - by Pocket Soft	195	189
TLIB	100	89
Editors		
BRIEF	195	Call
Edix	195	159

Editors Cont.	List	Ours
Epsilon - like EMACS	195	169
KEDIT - like XEDIT. V/4	150	125
SPF/PC - V2.0	245	185
Vedit Plus	185	129
File Addons		
Btrieve ISAM - V5.0	245	185
Btrieve/N - multiuser	595	469
Xtrieve + - by Novell	595	479
c-tree by Faircom - source	395	319
d-tree - data dictionary	495	399
r-tree - report generator	295	245
e-tree - w/r-tree	650	519
CBTREE - Source	159	141
CQL - full w/ source	395	349
dBC ISAM III - by Lattice	250	199
db_FILE - network model	Varies	Call
db_RETRIEVE - SQL	Varies	Call
XQL-SQL by Btrieve	795	649
UserSoft/C	299	229
WKS Library	195	189
FORTRAN		
FOR_C - FORTRAN 77 to C	750	679
Lahey FORTRAN F77L	477	439
Lahet Personal FORTRAN	95	89
MS FORTRAN - CodeView	450	299
Spindrift - DOS, windows	149	135
General Addons		
C + O Class Library	195	179
C Tools Plus - V5.0	129	109
C Utilities - by South Mountain	185	149
Greenleaf Functions	209	159
Greenleaf SuperFuntions	265	209
Turbo C Tools by Blaise	154	134
Graphic Addons		
GraphiC	395	319
Greenleaf Makeform-DOS	170	99
GSS Dev't Toolkit	595	519
Halo '88 - 140 + devices	325	245
Hoops 3D Graphics	575	499
MetaWINDOW	195	159
MetaWINDOW/PLUS	275	229
PCX Programmer's Toolkit	125	115
Pizazz Plus	149	129
Object-Oriented		
Actor	495	439
C Talk	150	139
C Talk/views	450	429
Smalltalk/V	100	85
Zortech C ++	150	129

OS - Support	List	Ours
DESQview	130	115
MKS Toolkit - Unix shell	199	179
MS Windows/286	99	75
MS Windows/386	195	149
Development Toolkit	500	379
Omni View	80	69
Other Languages		
Modula-2 Dev. System- by Logitech	249	199
RPG II complete	1400	1299
TopSpeed MODULA-2 Compiler	100	89
Other Products		
Carbon Copy Plus	195	159
CO/SESSION	249	219
Duplicator Toolkit	80	75
FM-Plus	100	89
Link & Locate ++ - ROM MSC	395	359
Math Advantage	495	459
Norton Utilities Advanced	150	109
PC Anywhere III	145	129
PC Tools Deluxe	80	75
PC Kwik Power Pack	130	119
QEMM-386	60	55
Remote2	195	139
Resident C by South Mountain w/source	99	85
SpinRite	198	159
VTerm220	59	55
Turbo Pascal	245	219
graphics-Menu	149	139
Say What	50	49
Topaz	79	69
TP2C	249	229
Turbo Analyst	99	89
Turbo ASYNCH PLUS	129	99
Turbo Pascal 5.0 by Borland	150	119
Turbo POWER TOOLS PLUS	149	129
Turbo Professional	125	109
Turbo Programmer	549	449
Text Screen Addons		
AEWindows	225	209
C-scape	399	379
C Windows Toolkit	100	95
C Worthy w/forms	295	Call
Curses - by Aspen Scientific	159	139
Greenleaf DataWindows	295	229
Hi-Screen XL	149	129
Hyperinterface- TFN, C, PAS menus	100	89
JAM by JYACC	595	469
MEWEL	150	139
POWERSCREEN by Blaise	129	99
Vitamin C - source, menus	225	169
VC Screen - painter	149	119
Vermont Views- replace WFD	395	Call
Unix/Xenix		
DosMerge 286 (unlimited)	249	229
DosMerge 386 (unlimited)	499	479
Edix - editor	275	239
JAM JYACC Applic. Mgr.	many	Call
Micorport System V/286 (ltd.)	649	599
Microport System V/386	899	799
SCO Xenix System V/286	1295	999
SCO Xenix System V/386	1595	1299

Note: Mention this ad. Some prices are specials. Ask about COD and POs. Formats 3" laptop now available, plus 200 others. UPS surface shipping $4/per normal item. Prices subject to change without notice.

We carry over 2,000 *different* products. Call today for complete technical information and advice, our free catalog, literature and *solid value.*

Canada 800-446-3846
Mass: 617-740-2510
Telex 671-5348
FAX: 749-2018

800-421-8006

HOURS:
M-F 8:30-8:00, Sat. 10:00-4:00
Eastern Standard Time

5 Pond Park Road
Hingahm, MA 02043

Number 114 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

right: To write the best code when nothing less will do, the best programmers use assembly language. And you know, I suspect that's exactly the sort of thing people read Programmer's Journal *for. Anyone can pick up a graphics book and write a tolerable line-drawing routine in C. Where do you look to get a fast implementation, though? I'd like to think the answer is* right here.

Anyway, reasonable people may disagree. John, thanks for (as always) intelligent and thought-provoking feedback.

Just Ask

I enjoy Michael Abrash's "On Graphics" series very much. His articles are clear, well structured, and fun to read. He gives a very good insight into the interior of graphics cards and their programming.

To my point of view, the description of higher 256-color resolution on the VGA *[PJ 7.1]* was one of his most exciting articles. Unfortunately, he restricted the program examples to the 320x400 mode, though 360x480 256-color mode seems to be much more interesting. I (and probably a lot of readers) would be glad to get further information about this mode in one of the following issues of *Programmer's Journal*.

Martin Althaus
Schwelm, West Germany

Michael provides just what you're looking for in his article in this issue. Readers may also be interested in Michael's new book, Power Graphics Programming, *a collection of the entire "On Graphics" series to date.*

Even More on Michael

I read with great interest Michael Abrash's "Measuring Performance" piece in the July/August issue. Several years ago I read the original *BYTE* article mentioned by Abrash, which described the method to generate high-resolution timings, and I wrote a simple package for my own use. This eventually evolved into a Shareware-distributed package that provides a multitude of high-resolution timer functions and supports popular C and Pascal compilers, with the current version (PCHRT v3.00) capable of timing hardware and BIOS interrupt service routines. I thought a few observations regarding this article might be helpful to those readers wishing to build on the routines listed.

The range of the timer can be expanded past 54 milliseconds by leaving interrupts enabled during the event to be timed and reading the DOS time-of-day data word as well as the 8253 timer chip. This alleviates the need to reset the timer chip before each timing and allows the system clock to function normally, while yielding a timer range measured in hours with microsecond accuracy. The design of a suitable data structure to hold these high-resolution time stamps can give the user the ability to have many different event timers running concurrently.

The coding of these timer routines need not be done in assembly language, as the majority of the popular high-level languages contain some type of function to read and write PC I/O ports. I have enclosed some simple Turbo Pascal code to illustrate how this might be done. The calculation of the elapsed time between two time stamps and the calibration of the timer system is beyond the scope of this letter but is not very difficult.

Two final comments: I have not found the delays Abrash used between 8253 chip I/O to be necessary, and, like the author of the original *BYTE* article, he never sets the 8253 timer chip back to its original state when all timing activity is concluded. I have rectified this in my sample code *[see the box above]*.

Thomas Leathley
Mt. Pleasant, Michigan

Something to C

Thank you for providing us with an opportunity to respond to Rex Jaeschke's article, "Lint Utilities" [*PJ* 7.4], where he summarized the history of Lint and reviewed Gimpel's PC-Lint and our C Programmer's Toolbox Volume II's CLint.

By and large, we agreed with Rex's review but were surprised that he did not also include a comparison/description of UNIX's Lint, since it is the granddaddy of all Lints. We agree that Lint and other supporting tools are extremely important in the software-development process: They save time and improve program quality. We also agree that documentation is extremely important. Unfortunately, as he noted, we did not update CLint's printed documentation in our last release—only the on-line documentation.

Though Rex seemed to advocate extensive documentation, including program-generated error messages, I personally believe that short and to the point is better. Who wants to wade through many pages of documentation before you can figure out how to operate a tool? (Yes, I know we have over 400 pages in our manuals, but you can do a lot simply with our on-line documentation.) Or how about wading through pages of warning/error messages

Code from Thomas Leathley's letter

```
Turbo Pascal high-resolution timer example. This timer reads the DOS time-of-
day data word and thus extends the range of the timer beyond 54 milliseconds.
Interrupts must be enabled during timing. The calibration routine, as well
as the routine to compute elapsed time between two high-resolution timestamps,
is left to the reader.

tphrt_type = record
               ticks : word;              { 838 ns 8253 timer ticks     }
               tocks : word;              { BIOS TOD 54.925 timer tocks }
             end;

procedure t_start;
{-----------------------------------------------------------------------------
 |  This procedure initializes the PC 8253 timer chip to mode 2 and calls    |
 |  the timer-calibration routine.                                           |
 -----------------------------------------------------------------------------}
begin
    inline(
            $B0/$34/                    { mov  al,00110100b }
            $E6/$43/                    { out  43h,al       }
            $2B/$C0/                    { sub  ax,ax        }
            $E6/$40/                    { out  40h,al       }
            $E6/$40                     { out  40h,al       }
          );
    t_calib;                            { call the calibration routine }
end;  { t_start }

procedure t_get(var timestamp : tphrt_type);
{-----------------------------------------------------------------------------
 |  This procedure grabs the current counts from the 8253 timer and the      |
 |  BIOS time-of-day clock. Interrupts are off during the timer reads,       |
 |  then back on.                                                            |
 -----------------------------------------------------------------------------}
var
    bios_tod : word absolute $0040:$006C;
begin
    inline($FA);                        { interrupts off }
    port[$43] := 0;
    timestamp.ticks := port[$40];
    timestamp.ticks := timestamp.ticks + (port[$40] shl 8);
    timestamp.tocks := bios_tod;
    inline($FB);                        { interrupts on }
end;  { t_get }

procedure t_stop;
{-----------------------------------------------------------------------------
 |  This procedure disables 8253 mode 2 and sets 8253 mode 3 counting,       |
 |  which is what the 8253 was doing before we meddled with it. . . .        |
 |  Invoke when all timings complete.                                        |
 -----------------------------------------------------------------------------}
begin
    inline(
            $B8/$36/                    { mov ax,00110110b }
            $E7/$43/                    { out 43h,ax       }
            $2B/$C0/                    { sub ax,ax        }
            $E6/$40/                    { out 40h,al       }
            $E6/$40                     { out 40h,al       }
          );
end;  { t_stop }
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

No Other MS-DOS Linker...

Helps you overcome the 640K barrier with a virtually unlimited and flexible overlay manager,

and....

helps you improve program execution speed,

and....

overcomes the large program "compiler fast but linker slow" problem

and....

prevents duplicate code in multiple EXE systems from chewing up disk space on development machine, shipping media, and target machine,

and....

provides in-depth, highly detailed information on interrelationships and dependencies of all modules of your program,

and....

works with almost all MS-DOS compilers.

Powerful Overlay Manager

RTLink supports two separate overlay schemes.

The first of these schemes parallels the method used in the Microsoft linker. It supports one overlay area and must have language library support.

The second overlay scheme which **RTLink** provides is much more flexible. It allows up to 64,000 overlay areas to be interspersed at separated or contiguous locations within the memory image. Each overlay area can in turn contain smaller, nested overlay areas within it which can differ depending on which primary overlay is in memory at the time. The portions of code which are brought into each overlay area (called overlay sections) can be placed into either the executable (.EXE) file or into separate, external files whose names you specify. There can be up to 64,000 overlay sections. You can specify that a specific overlay section be loaded into a specific overlay area when the program is brought into memory during program startup. You can specify the code which is to be a part of a specific overlay section by naming either object files, modules, classes, or groups. There is no overlay interrupt to conflict with the other interrupts on an end user's system. This second scheme requires no language library support and is language independent. It represents the state-of-the-art in overlay technology.

Fast Development Linking

Run-Time Libraries (.RTLs) are optional, easily-created files that can be used to manipulate link speeds. During development, RTLs can be used to store pre-linked modules. This allows programmers to set aside much of the code that is not being modified, thereby significantly enhancing link speeds. In straight links, RTLink is already faster than your current overlay linker. However, RTLink is even faster when RTLs are used as a productivity tool during the development and testing phase.

Profiling

RTLink comes with a built-in performance analyzer. This profiler provides highly-detailed performance statistics about your program. Unlike other performance analyzers, these statistics are obtainable without code modification at user-determined/machine-capable timing sequences adjustable to thousandths of a second. Source code is provided for a read/analysis utility which can be used to customize the statistics reports. The statistics provided establish where the time was spent during the program execution and what calls were made as the program executed.

Significant Disk Space Savings

RTLs can significantly reduce the amount of disk space needed to store and/or ship your programs if you have multiple EXE files. When multiple EXEs are present, much of the code (especially language library and certain utility code) is commonly used in each EXE. By placing this redundant code and/or data into a RTL(s) that can be shared by each EXE, you eliminate duplicate storage - which saves disk space.

Interrelationships of Program Modules

The LIBLIST utility provided with **RTLink** can be used to obtain in-depth, highly-detailed information about the interrelationships and dependencies between all modules of a program. This information is useful in overlay design as well as creation and maintenance of run-time libraries.

- **Works with almost all MS-DOS compilers.**
- **No additional memory utilized**
- **No source code changes**
- **No charge for Tech Support**

Order yours today for $195.00

Phone in your order using Visa, MasterCard, or C.O.D., or send a check for $195.00 to:

For Phone Orders Call 800-456-7032

FAX (713) 460-2651

Pocket Soft, Inc.
7676 Hillmont, Suite 195
Houston, Texas 77040
(713) 460-5600

Number **279** on the Reader Service Card

NOTE; Texas Residents include 7% sales tax

Trademarks: All Pocket Soft Products are trademarks of Pocket Soft, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders. ©1989 Pocket Soft Inc.

PCMAGAZINE LICENSED MATERIAL
ELECTRONIC REPRODUCTION PROHIBITED

that have similar, lengthy descriptions? When an error condition occurs, the exact error location (file, line, *and* character) should be identified and a self-explanatory error message generated. Certainly, there will be error messages or conditions that require more-complete descriptions than can be incorporated in a program-generated error message. These exceptional conditions should be documented in the manual with possible workarounds or solutions.

Finally, I would like to mention that in the middle of September, we will be releasing the next version of the C Programmer's Toolbox (Rev2.0). The release includes all-new documentation (which should satisfy most of Rex's complaints) and a number of significant tool enhancements. Some of CLint's enhancements include selective enabling/disabling of error messages, more control over **typedef** checking, and support for a number of nonstandard (and undocumented) compiler features (e.g., "//"). These new enhancements complement our continuing support for ANSI C, Microsoft's C, and Borland's C, as well as for many of the engineering-workstation Cs.

Other major tool enhancements include virtual-memory support for CFlow (program hierarchy) and CXref (cross reference), now enabling quick and easy processing of *unlimited* program sizes (the symbol tables now automatically overflow to disk); more options and controls for CPrint, our C source code beautifier/reformatter; and more options in PMon (performance monitor) for intercepting and monitoring a program's external environment, including BIOS. In addition, many of the other supporting tools have also had significant enhancements. With this release, many more of the tools are now the best available on any development platform.

We wish to thank all of those people who have worked with us (including [*PJ* Editor M. Steven Baker] on CFlow) and helped us to make our tools the best and most useful available. We will continue to strive to provide the best possible C development environment.

W. T. Sumerlin, Jr.
MMC AD Systems
Milpitas, California

Getting the Time of Day

I got my copy of the July/August *PJ* today, and lo! not one, but two, articles on my favorite topic of timing. Each article contains half of the answer to getting 840-ns.-resolution timing up to about 138 days, more or less, if my memory serves me well. So, let's put it all together.

Think of the time of day as a 48-bit number, in which the high-order 32 bits are the time of day as maintained by BIOS, and the low-order 16 bits consist of the current count read from 8253 timer 0. So, reprogram 8253 timer 0 in mode 2 per Michael Abrash's approach. Then, to get a current time value, inhibit interrupts, snatch the current BIOS clock value (as Ethan Winer did), and read in the current 16 bits from timer 0.

Voila! Put it all together, and you have 48-bit time of day accurate to 840 ns., the resolution of the 8253. Or almost. The 16 bits of the 8253 must be inverted, and so they are monotonically ascending, like the rest of the time of day. Now, save the 48-bit time value, and enable interrupts. Inhibiting interrupts guarantees that there will be not be a timer interrupt while putting together the time-of-day value. Occasionally, there will be a lost BIOS tick in this whole process, as there always is in DOS when the timer interrupt cannot be serviced because of servicing a higher-priority interrupt. More often than not, a BIOS tick is lost when there is either a disk I/O with a large sector count in process when the timer counts to zero or a lengthy handshake between the PC and a LAN card.

Now, to find the elapsed time between any two time-of-day readings, do 48-bit subtraction on the two numbers. Very accurate, and the system clock isn't disturbed by frequently resetting the 8253. If

New Rev2.0 Release & New Products

The C Programmer's Toolbox contains more than 20 tools for the serious C programmer. The tools are now available in three different development environments: PC/MS DOS; Macintosh's MPW *(new)*; and Sun's UNIX *(new)*. Some of the major tools include:

CDecl	Translates C declaration statements to/from English *(helps you decipher those complicated C variable declarations and construct new ones). {MPW & Sun only}*
CFlow™	Determines how a program is organized and what is in the runtime library *(quickly shows you function hierarchy and interdependancies)*
CHilite™	Highlights and prints your C source code *(great for creating your final documentation). {MPW & Sun only}*
CLint™	Syntactically checks one or more of your C source code files *(use this to find those silly, syntactically correct programming mistakes)*
Cpp	Preprocesses C source code files that have ANSI macro definitions *(helps you port ANSI C code to/from your target environment). {MPW & Sun only}*
CPrint™	Reformats/beautifies your C source code *(makes code easy to read and ends disputes on source code formats).*
CritPath™	Determine a program's critical path. *(could help you improve your program's performance) {PC DOS only}*
CXref™	Cross references and checks usage of up to 17 different classes of symbols in your C source code *(through its virtual memory system you can cross reference any size program or number of listings)*
PMon™	Monitors a program's execution *(non-invasively monitors your program's function execution, system service and BIOS usage). {PC DOS only}*

C™ Programmer's Toolbox / MPW PC Sun

The tools include support for ANSI C, Microsoft C, Turbo C, Quick C, Sun UNIX C, Macintosh's MPW C, LightspeedC and more.

This new Toolbox release easily handles any size program through its new virtual memory system. Also, several tools include a new C macro processor that accepts K&R, ANSI and other C dialects.

C Programmer's Toolbox™
MPW ($295), PC Volume I or II ($99.95),
PC Volume I & II ($175), Sun ($495)
C Programmer's McTool Series™ (Macintosh - non MPW)
McCPrint™ ($59.95), McCLint ($99.95)

Call and Order today...
(CA residents, please add sales tax; Overseas shipping extra)

MasterCard VISA®

MMC AD Systems
Box 360845
Milpitas, California 95035
(408) 263-0781

Number 94 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the result of subtracting the earlier from the later time value is negative, you need to add a correction factor of at least one BIOS tick to compensate. Then, keep track of the number of lost ticks, and apply a correction factor equal to *n* lost ticks.

I worked this out from the Intel 8253 spec several years ago and then corroborated it with an article by Ralph Brickner in *PC Tech Journal* from back in 1984. Ralph did a two-article set. The first described a home-grown profiler, and the second reviewed the state of the art at the time. We've used the algorithm described above in one commercial product and are about to incorporate it into a second. The only fly in the ointment is interpretive BASIC, which reprograms timer 0 itself to count down four times per 54.925 ms. So, you can't get accurate timings of interpretive BASIC. Oh, well.

Rick Naro, the designer of the Inside! profiler series, used roughly the same approach to do timings. By now, Michael Abrash probably has used one version or another of Inside!, which Ethan Winer reviewed very favorably in *PC Magazine*, and rightfully so.

I am amazed at whatever thoughts were in the mind of the engineer who wrote the 8253 timer routines for the original PC BIOS. I'll never understand why mode 3 was chosen instead of mode 2!

Overall, this was a good issue, full of interesting ideas and topics.

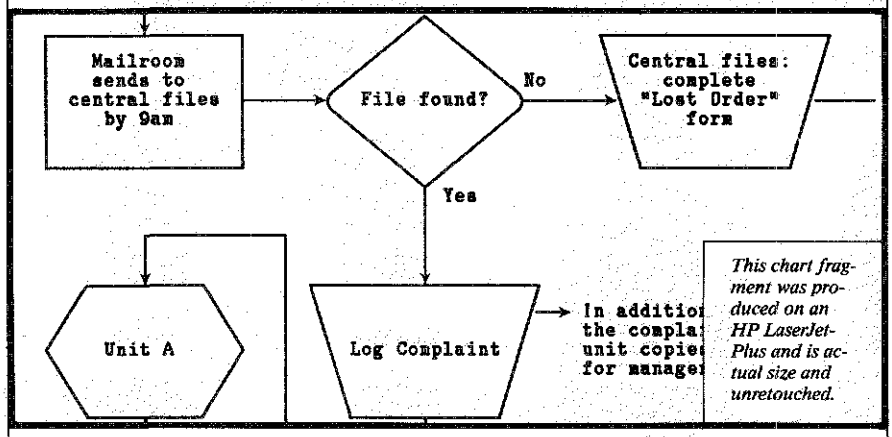
Ben Myers
Spirit of Performance, Inc.
Harvard, Massachusetts

MICHAEL ABRASH RESPONDS: Great feedback. Actually, I'm well aware of how to time periods greater than 54 ms, but (a) precision is lost if interrupts are turned back on, and (b) there's only so much room in my column in PJ. *And, yes, Inside! is very good (I discuss it in this next issue). I must say that I think that a 32-bit timer tick count is sufficient (once you're over a minute you might as well forget about messing with the 8253!). Also, there is a bit of a synchronization problem with reading the 8253 and the timer count simultaneously: It can introduce 54 ms of inaccuracy.*

We at *Programmer's Journal* like to hear from our readers. Please address correspondence to Managing Editor, *Programmer's Journal*, 150 N. 4th St., Springfield, OR 97477. You can also contact us via fax (503-746-0071), CompuServe (user ID 76676,427), or MCI Mail (user ID 336-4248). All letters become the property of *Programmer's Journal* and may be edited for publication.

EasyFlow

An on-screen chart processor that knows about charts and diagrams, not just another "screen-draw" program that makes you do most of the work. **EasyFlow** is a powerful full-screen graphics program dedicated to data-flow diagrams, flow charts and organization charts. With it you can quickly compose diagrams. More important, you can easily modify them so they are always up to date.



This chart fragment was produced on an HP LaserJet-Plus and is actual size and unretouched.

Automatic. Fully automatic text centering within shapes. Fully automatic line routing & re-routing. Fully automatic placement of text along lines.

Fast. Written in assembly language for speed.

Large. Diagram size limited only by available memory. Supports LIM expanded memory. Diagram too large for your printer? EasyFlow automatically breaks the diagram up & prints it in page size pieces.

Standard. All standard data flow and flowcharting shapes included. Other shape libraries are available.

User friendly. Don't take our word for it. PC Magazine says *"EASYFLOW lives up to its name. It's hard to imagine any easier and more flexible way to produce basic and even complex flowcharts"*.

Mouse. Optional but fully supported.

It works. Over 50,000 copies sold. Two of the "big eight" accounting firms have world wide site licenses for the use of EasyFlow.

It prints. On most popular printers including IBM, Epson, Toshiba, HP LaserJets, HP Desk Jet XEROX 4045, postscript printers and many others.

It plots. On HP7440, 7475, 7550 and 7585B plotters.

Documented: 130 page manual plus over 150 screens of context sensitive help.

Rush delivery: Order by noon today (eastern time) and you'll have it tomorrow. Rush delivery charge is $10.50 (instead of $2.00) and is available only to continental USA and Canada.

EasyFlow works on IBM PC's, IBM PS/2 and compatibles. Requires 512K memory, hard disk and an IBM CGA/EGA/VGA or Hercules monochrome compatible adapter card.

Order direct for only **$149.95** + $2.00 S&H (USA/Canada), $10.00 (foreign). Payment by M.O., check, VISA MasterCard or Company PO.

Haven Tree Software Limited
PO Box 1093-V
Thousand Island Park, NY 13692

HAVENTREE SOFTWARE LIMITED

Order Desk: 1-800-267-0668
Information: (613) 544-6035 ext. 31
Telefax(G3): (613) 544-9632

Number 154 on the Reader service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A Technological Schism

by

Hal W. Hardenbergh

Most of us who read the newspapers know that the Moslem world is mostly divided into two sects, Sunnites and Shiites. I once read that the schism was based on a dispute over the ownership of a garden—not the Garden of Eden, but just a little garden like one any suburbanite might have. Because a guy named Martin Luther didn't like certain then-prevailing practices of the Catholic church, I attended a Protestant Sunday school as a child.

I don't think anybody really cares anymore about a dinky little garden, but you do not see the Sunnites and Shiites reuniting. I suspect that Martin Luther might find today's Catholic practices more to his liking were he still around, but we do not see the Protestants and Catholics reuniting.

Like some others, I changed religions when I grew up. For many years now, I have venerated the electron, both in its manifestation as a particle and in its manifestation as a wave. Last year I moved to Silicon Valley, where many of my fellow votaries are located.

Regrettably, my religion has in recent years suffered a schism between us steadfast CISC devotees of truth and the misguided followers of the false heresy, RISC. My purpose in writing this article is to briefly review the history of this schism and to provide an unbiased description of its future course.

It was all Ivan Sutherland's fault. He introduced Lynn Conway, who worked for Ivan's brother Bert at Xerox PARC, to Cal Tech's Carver Mead. Conway and Mead developed a systematic way to design VLSI circuits and more or less invented the concept of the silicon foundry. Designing VLSI chips became so easy, and fabricating them became so cheap, that even a college class could design one and have it built.

Conway and Mead's book was published in 1980. The next year, UC Berkeley's David Patterson had his class design a computer chip. Even with Conway and Mead's techniques, that computer had to be simple, since a class does not have years to work with. So, they reduced the instruction set to fit the time constraints—and they left out a lot of useful stuff, such as an interrupt handler.

Because of the extreme simplicity of the chip, they were able to make it 32 bits wide. Remember, this was 1981. The newest and hottest chip in the commercial world was the 68000, which had only a 16-bit data bus. Bus bandwidth counts. The simple chip was completed in 1982, and it worked! That it was fast was perhaps an unexpected bonus.

This chip, which Patterson called a RISC, was deservedly considered a success. A college class had produced a real, working microcomputer chip that could do loads and stores and Boolean operations and shifts and conditional branches. Students were jumping up and down and (this being the '80s) exchanging high fives. *They* could build a CPU!

RISC wasn't really all that new. Other folks had worked on the concept, including John Cocke in Building 801 at IBM's Thomas Watson Research Center, in Yorktown Heights, New York, in the mid '70s. (Some people think that Seymour Cray's designs, such as the 1964 CDC 6600, are actually RISC computers.) But Cocke didn't talk to reporters, and Patterson did; so most folks associate Patterson with RISC.

Patterson repeated his success with another class. Now there were *lots* of students jumping up and down. In the natural course of time, these students graduated and diffused into industry.

The next thing you knew, announcements of wonderful new commercial RISC chips, all with 32-bit data buses, began to appear. And *much* later, some of them actually became real. (Some of them are *still* in the process of becoming real.) Because they had 32-bit buses, they were all quite fast. But a funny thing happened in the years that had passed: Motorola and then Intel switched their CISC designs to 32-bit data buses, and suddenly the RISC supporters noticed the RISC performance advantage was evaporating.

(The AMD 29000, which is just becoming real, *should* be a lot faster than the 68030, which will soon be replaced by the much-faster 040. But it isn't. The 030 does integer multiplies 45 percent faster than the 29K! A new CPU should be twice as fast as one that is two years older, or the designers aren't doing their job.)

In the past couple of years, commercial microcomputers based on RISC microprocessors have appeared on the scene from Hewlett-Packard, MIPS, and Sun. A very peculiar thing happened: RISC technology was expected to be cheap because the CPUs are simple and hence small. But those first commercial RISC microcomputers wore price tags ranging from $76,000 to $260,000. Why?

Nineteen eighty-seven was a terrific year for personal computer types. We had enough memory (typically a megabyte), and that memory was fast enough for our CISC processors. But 1987 was a mixed year for the folks who were rolling out the first RISC microcomputers. RISC uses many 32-bit instructions, which must operate very quickly. This means that RISC machines need twice as much memory as CISC machines to hold the same program. And they all ran UNIX. In 1987, UNIX had been "improved" to the point that even CISC-based machines, such as the Sun 3/260, needed 4 megabytes to get any useful work done. That meant the RISC machines needed 8 megabytes. And the RISC

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A review: Semiconductor memory capacity is doubling every 1.5 years, as it has for the last 20 years. CPU speed is doubling every 2 years, as it has for the last 42 years. Memory bus bandwidth is doubling every 3 years, as it has for the last 20 years. (By *bandwidth* I mean the ability of the memory system to get data to the CPU fast enough to keep the CPU running at full speed. This is sometimes called *latency*.)

These are all exponential increases, which plot as straight lines if we use a logarithmic vertical axis and a linear horizontal time axis. But the slopes are different, and that means the lines cross someplace, as shown in the figure, which is normalized for constant CPU performance. Ten years ago we had memory speed to burn but were really hurting for memory capacity.

machines needed to fetch all those instructions very quickly, which meant they needed memory that was twice as fast as that of an equivalent CISC machine. In other words, all of the RISC machines needed very large, very fast memory systems.

Do you suppose that we have discovered why RISC-based microcomputers were being sold for $76,000 to $260,000 when *excellent* 386-based machines, comparably equipped except for that excessively large and fast memory, could be had for $10,000?

Why, you may ask, did Cocke and Patterson settle on RISC if it has substantial memory-speed disadvantages? Well, in 1975 and again in 1981 the memory-bandwidth problem that plagues today's RISC machines did not exist (check the graph). The memory-size problem, which is a two-to-one hit for RISC, was not really a problem in the 1975 IBM environment, where low markups did not prevail. And Patterson was developing a CPU chip, not a complete computer. *What* memory problem?

That brings us almost up to date. What's happening now? Well, Intel has announced both the 860 and the 486, which are RISC and CISC chips, respectively. The 486 represents everything that is evil to the unreasoning RISC camp followers. Why? Well, RISC chips always execute fundamental operations—load and store, for example—in fewer clock cycles than CISC machines, such as the Intel *X*86 series. Don't they?

Ahem. The SPARC requires two clocks to load and three to store. The 486 requires one clock to load and one to store. And they both run at 33 MHz. *What* inherent RISC superiority? In fact, which is the RISC and which is the CISC? Aren't RISC chips fast load–store devices?

My article "Memory Trends" was published in *PJ* 6.3 (May/June 1988). While it was being written, Cypress Semiconductor, a maker of fast static RAM, had 25 percent of its fabs sitting idle. Six months later, the fabs were fully booked and the backlog was growing. Why? Because the use of high-speed static RAM in personal computer caches skyrocketed. Who woulda thunk it? Oh, yes—that article ended with the following sentence: "You are going to see true 64-bit microprocessors *soon*."

Such coincidences. The 860 just happens to have a 64-bit bus. It has an internal data cache with a 128-bit bus and an internal instruction cache with a 64-bit bus. That makes an internal cache bus width of 192 bits! For that reason, the 860 is today's fastest microprocessor. Too bad it won't run MS-DOS.

Lucky guesses? No, I was just extrapolating from long-established trends. One predicts a deviation from these trends at one's peril. It's been *two* years since the 10-MIPS SPARC was announced, and Sun isn't shipping 40-MIPS versions yet. Sun was going to double SPARC's performance *every* year, remember?

Just *What* Is a RISC Chip?

In 1982, a RISC chip was simple, potentially cheap, and faster than then-current CISC chips, for the reasons discussed earlier.

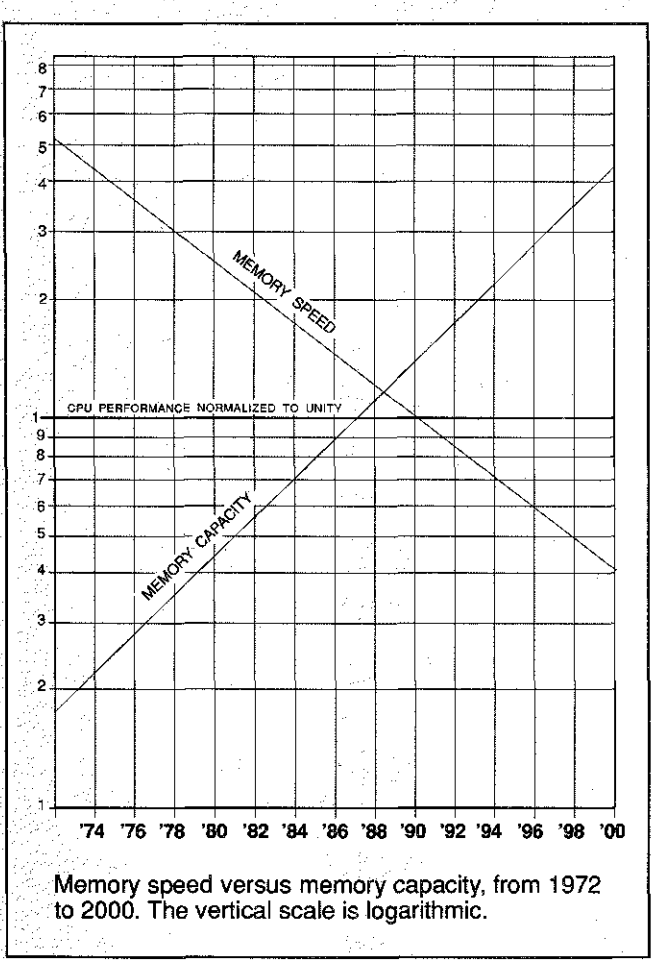
Today, a 32-bit RISC chip is neither simpler, cheaper, nor faster than its CISC counterparts. The disputed garden that originally caused the RISC–CISC schism did in fact exist in 1982 but has now vanished. However, religious schisms historically are not "cured" when the original causes vanish. There is an *enormous* number of persons and companies out there who have a vested interest in RISC.

All of us know that minicomputers are faster than personal computers and that mainframes are faster than minicomputers. In past years, that statement could be made with confidence; today one squirms a bit—the lines are blurring. In

Transistors get smaller and hence faster as time passes. But both memories and CPUs use the same transistor technology; so this has nothing to do with the relative speed of memories and CPUs. Let us hypothetically assert that transistor speed is constant.

As time passes, memories get bigger. That means that they get slower, because the address decoder has more gate delays. As time passes, CPUs get bigger, which means they get faster. The disparate trends of memory and CPU speeds will continue, because both memory and CPUs are going to continue to get bigger (especially in transistor count).

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Memory speed versus memory capacity, from 1972 to 2000. The vertical scale is logarithmic.

recent years, a microcomputer was something made from one or two commercial chips, and a minicomputer was made from five or six custom gate arrays.

The Cypress SPARC chip set has five or six chips. The Edgecore (formerly Edge) implementation of the 680X0 instruction set has five or six chips. The upcoming Nexgen implementation of the 386 instruction set has five or six chips. Are these microprocessors or minicomputers? You tell me.

Motorola is vanishing from desktops. Sun has just now released its first 33-MHz 030 machine, for $40,000. H-P has been selling its 33-MHz 030 machine for $44,000. Motorola makes all the 50-MHz 030 chips anyone would want, and *nobody* is shipping a desktop computer based on that part. Apple's 020 and 030 offerings are ludicrously slow, perhaps because Apple has a monopoly on the Mac OS. Meanwhile, three companies announce 33-MHz 386 machines, and reviews of these machines appear in magazines, *before* Intel starts shipping the parts!

Motorola's 50-MHz 68030 screams, and the forthcoming 68040 will be to the 486 what the 030 was to the 386. But I greatly fear that I will never be able to purchase a full-speed 030 or 040 machine to take home. I *know* that I will be able to do that with a full-speed 486 machine, and *soon!* Maybe it's time for me to start learning Intel assembly language....

Intel's *X*86 series has vanished from embedded applications, which do not have to run 1-2-3 but do have to be easily programmed. The 32-bit microprocessor market was $1.1 billion in 1988. Intel and Motorola each had 43 percent, National's CISC chips had another 5 percent, and everybody else had 9 percent (Integrated Circuit Engineering Corp., Scottsdale, Arizona). "Everybody else" includes CISC makers Zilog (80000), AT&T (32100), and various Japanese CISC TRON chips. I'm looking hard to find the RISC share of the market, but it's apparently hiding under a speck of dust.

Right now, Intel has the world's fastest microprocessor, the 860. It also has the world's fastest 32-bit microprocessor, the 486. There are rumblings of a new version of the 960 that will give the 860 a run for its money. How is this possible? In the first quarter of calendar 1989, Intel spent $83.2 million on research and development—that's $665 million per generation of CPUs. Intel can afford to develop *several* high-performance CPUs concurrently, even if the cost per CPU exceeds $100 million.

Want to buy stock in a RISC startup?

A Look at the Future

The 68040 will be the last of the 32-bit micros that have any pretention of high performance. The 586 and 68050 will both have 64-bit data buses. Look for the first 128-bit micro in 1994 and the first 256-bit micro in 1999.

Workstation manufacturers have begun to notice the jillions of 386 machines running AutoCAD, OrCAD, and 1-2-3. If the thought of those packages running on 486 machines at under $10,000 doesn't scare them, it should. A mini-war has just started over low-end UNIX machines. So far, DEC, Sun, and Data General are slugging it out. *All* of the workstation manufacturers are going to lose until they notice that *X*86 machines are sold across a retail countertop by salespersons who are behind on the payments on their 1978 Fords. Workstation salespersons drive new Porsches, belong to country clubs, and send their kids to private universities. That's why you can buy a 33-MHz 386 personal computer for under $10,000 but you have to pay $40,000 for a 33-MHz 68030 machine from a workstation vendor. (The cheapie UNIX stations you read about are diskless, and so they are totally useless as personal computers.)

Has anybody noticed that younger Compaq (1988 profits: $255 million) is growing a lot faster than Sun Microsystems (1988 profits: $66 million)?

There Was No Choice!

YEAR	STATIC		DYNAMIC	
1971	??	(256x1)	1103	(1Kx1)
1974	2102	(1Kx1)	??	(4Kx1)
1977	2114	(1Kx4)	4116	(16Kx1)
1980	6116	(2Kx8)	4164	(64Kx1)
1983	6264	(8Kx8)	41256	(256Kx1)

In past years, the CPU determined the system speed. So there were lots of CPUs to choose from, but just one static and one dynamic RAM. In future years, the memory will determine the system speed. There will be many memory variations, but only two CPUs will be used: the *X*86 and the 680*X*0.

Let me make this clear: You will buy computer performance the same way you buy steak—you get more if you pay more. Fancy CRTs or fancy cuts are more expensive. There will be a market for $2,000 computers and $5,000 computers. The more expensive computer will *always* be faster, because more dollars buys faster RAM, larger RAM, or a little of both. CPUs will not be chosen for speed, because they will all be capable of outrunning the RAM. Instead, CPUs will be chosen to run a particular software base. And that means that all CPUs that are not code museums are going to vanish silently into the night.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Intel 586 will be announced about September 1991. It will have a 64-bit data bus and Harvard internal caches with 128-bit cache widths, and it will use 2 million to 2.2 million transistors. By December 1992, Intel will be using a 0.65-micron process to make this chip, and it will be running at 60 to 75 MHz. Cypress, which makes very fast static RAMs, has a profit margin of over 20 percent. Sun Microsystems, wildly successful vendor of UNIX systems, has a profit margin of 5 percent.

The perverse RISC fallacy was neither perverse nor a fallacy in 1981. Circumstances have dramatically changed since then. "RISCites" have invested tens of millions of dollars, years of their time, and untold quantities of personal pride and such into the RISC concept. Until recently, they cheerfully admitted that RISC devices require *more* memory and *faster* memory. Why do I expect this candor to wane?

A variety of CPUs will continue to be made—Texas Instruments will be selling the 4-bit TMS1000 into the next century—but only in embedded applications. There will be only two CPUs used in personal computers or personal workstations.

A $5,000 personal computer will have a big, slow memory system for folks who are databasing or a small, fast memory for computation-intensive applications. This is why you are going to see a variety of specialized RAMs. Not many personal computer owners are going to want to set aside 4 megabytes so that they can run UNIX or OS/2. MS-DOS will continue to be the most popular operating system because it is memory efficient.

Let me emphasize this: In past years, your only decision was how much memory to buy. In future years, you will have to decide both how big *and* how fast. This year, 80-nsec, 4-meg DRAMs; 35-nsec, 1-meg Hitachi DRAMs; and 20-nsec, 256K Toshiba static RAMs are all entering production and all wear essentially the same price tag. You will pay a 4*x* premium per bit for the 35-nsec part and a 16*x* premium per bit for the 20-nsec part.

Today, a complete 33-MHz 386 personal computer with a megabyte of fast DRAM and high-speed, static-cache RAM starts at about $8,000. I think it would cost at least $12,000 to build an equivalent machine based on the 50-MHz 68030. Strictly speaking, it will be possible in the near- and mid-term future to buy computers whose memory systems will run the CPU at full speed, but the price tag will continually increase. If we limit the price tag of a "personal" computer to, say, $10,000, then it is already (in 1989) not possible for memory to keep up with the 68030.

UN*X Report #37,617

Some of you know that I used to write a newsletter, parts of which were published in *PJ*. In late 1981, the "experts" were predicting that UNIX would soon take over the high-end personal computer market. Beginning in early 1982, I continually disputed that. I even correctly predicted that AT&T's UNIX PC would be a marketing disaster of major proportions. This is 1989, and UNIX still has 5 percent of the market. But hark! The *Wall Street Journal* published a report predicting that UNIX will have 38 percent of the market in 1993. Predictions of UNIX's imminent ascendency occur repeatedly, like sightings of Elvis or UFOs.

I now use UNIX at work. I document using **vi** and **nroff**. I test image-processing systems that run UNIX. I've learned the hard way that you can destroy the UNIX system disk data by turning off the power instead of following a lengthy software-shutdown procedure. The user interface is incredibly complex and unfriendly. *This* is a system that is going to take over the personal computing market? Running on a RISC CPU that requires an inordinate amount of the computer world's scarcest resource, memory bandwidth?

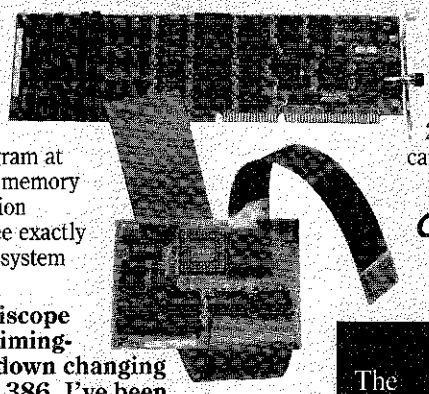
An Exercise for the Student

Plot the cost of a personal computer that would run the fastest then-available *X*86 CPU with no wait states over the past eight years. Now, project that curve forward over the next eight years. Interesting, isn't it? (A computer with cache does not really have *zero* wait states, but it's the best practical approximation to that ideal.) PJ

Hal W. Hardenbergh is a senior hardware engineer who sometimes programs. Hal works at Vicom Systems, Inc., in San Jose, California.

NEW FROM PERISCOPE® Real-Time Debugging For 286s and 386s up to 25MHz

Periscope® IV brings you a new generation of hardware-assisted debugging. It works on virtually any 286 or 386 with an AT-style bus running up to 25MHz with zero wait states. With Periscope IV, you can run your program at full speed while tracking down unwanted memory overwrites. And you can use the information captured in its real-time trace buffer to see exactly what the system is doing, and to improve system performance.



"The hardware really makes Periscope shine, especially when you've got timing-related problems. I can now track down changing pointers and altered buffers on my 386. I've been using it to debug Crosstalk® Mk. 4 and there's just no better way to do it," says user Jeff Garbers, Crosstalk Communications' Director of Software Development.

Call now for our FREE four-page Model IV brochure or to talk about your debugging needs. Prices range from $145 for just software to $2,995 for a Model IV for a 25MHz 80386 system. We accept major credit cards and qualified company purchase orders.

Order your Periscope, Toll-Free, Today! 800-722-7006

The Periscope Company, Inc.
1197 PEACHTREE ST.
PLAZA LEVEL
ATLANTA, GA 30361
404/875-8080
FAX 404/872-1973

Number 6 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Source Code Version Control

by
Ken Cunliffe

For the past year, I have managed the development of a new computer-aided design (CAD) system in the C programming language. Multi-CAD™, a part of XYSYS International's CAD-Access™ system, serves as a bridge between the PC and the mainframe CAD systems used by most large manufacturing companies. The system runs on IBM PC-ATs and compatibles and offers seamless data interchange between PCs and a wide range of mainframe CAD systems. The package is targeted at companies with mainframe CAD/CAM/CAE systems that wish to add inexpensive seats and also at companies not presently using CAD/CAM/CAE that need to interface with customers or suppliers who already have mainframe systems. The Multi-CAD source code includes hundreds of source files and hundreds of thousands of lines of code.

As sections of the program were constantly being passed from one programmer to another, one of the biggest problems we faced early in our development effort was making sure that everyone was working on the correct version of the code. For example, on several occasions, when one programmer went on vacation, other people on the project lost track of what the vacationer had been doing and incorporated an earlier version of the software into their work. While we tried to avoid having two people work simultaneously on one section of code, lead-time requirements sometimes made this necessary. The absolute worst-case scenario—the product goes out to the customer without part of an update—never happened, but we came close enough to make us very nervous.

Version-Management Control

After encountering serious coordination problems, we adopted a version-management tool, with excellent results. Version management seems to have been introduced with UNIX, and it has been one of the more useful tools supplied on UNIX systems. For those who are unfamiliar with the concept, a *version-management system* helps to manage the development and maintenance of software or other types of files by tracking changes made to those files. All changes are stored in an archive directory. Each time changes are made to a file, a new revision of the file is created, and the archives are updated. Only changed information is stored, helping to conserve disk space.

A set of changes that were made to a file to take it from one revision to another is referred to as a *delta*. Each revision is given a unique identifier, and so any revision of the file can be regenerated using its identifier. A group of file revisions that make up a version of a product is called a *thread*. The version-management system that we use automatically generates four predefined threads, called the *active, latest, test,* and *working threads*: The active thread of the product is made up of the currently extracted versions, the latest thread is the most recently created revision of each file, the test thread is the version of the product that is currently being tested, and the working thread is the current release of the product. The user is free to assign other threads as desired.

Here's how the version-management system we chose, the Seidl Version Manager (SVM), from Seidl Computer Engineering, solved the problem. Whenever a programmer edits a file, he or she must first "check out" that file from the version-management system. Rather than selecting from a mishmash of files on diskettes with cryptic identifiers, the programmer will, in most cases, simply tell the version-control system to select the latest thread. That revision is extracted from the archives and placed in the work directory by the version-management system. After editing the source file, the programmer "checks it in," or places it in the archives. During the update process, the version-management system computes the changes made from the base revision and creates a delta for this new revision. The version-management system ensures that everyone is working on the latest version of the program (unless we intentionally decide to do otherwise) and thus saves me a tremendous amount of time and worry.

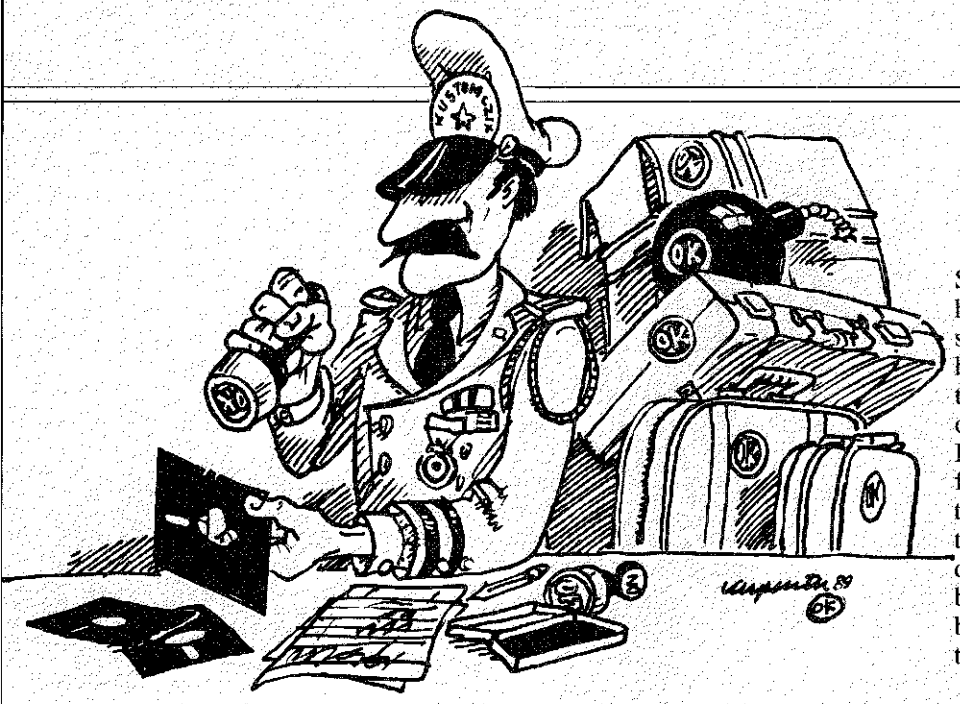
Simultaneous Development

Prior to installing version management, we also faced severe problems when we had to maintain simultaneous development efforts at certain stages of our programming effort. Parts of our program are required to run on Prime mainframe computers, at the request of a major customer. This client realized that time could be saved by off-loading some particularly CPU-time-intensive sections of the program from the PC. Because of time constraints, we simultaneously developed the Prime and PC versions of the program, incorporating the Prime environment changes into our main PC product and creating a central generic source code from which the Prime version can be regenerated quickly whenever the program is updated.

Program logic for the two systems is identical, but there are key differences in the file-handling systems. The main difference is that the Prime system inserts a counter at the beginning of each data record. Since this is the wrong structure for the PC that will be processing the data, it is necessary to turn off the counter facility for a direct-access file.

The adaptation for the Prime was the responsibility of one programmer.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



After a section of the program was running on the PC, he would take it, install it on the Prime, add the necessary code, and debug it. While this was going on, other programmers were making changes to the PC version; thus, it was necessary to manually merge the PC code with the revised Prime version—a task that was time-consuming and prone to error. This process frequently had to be performed at least twice for each section of code, because the PC sections were often revised enough to require revisions in the Prime code.

Version management greatly reduced the time required for the merging process described above. When a section of the code is near completion on the PC, we invoke the version manager's revision-branching feature to create two simultaneous lines of development (or threads, in version-management terminology). Working on one thread, our Prime programmer adds the necessary file-handling routines; working on the other thread, our PC programmer continues to fine-tune the program. As soon as the Prime programmer has completed his or her work, the version manager automatically merges the two development threads.

Occasionally, simultaneous changes may conflict. For example, one developer may delete lines 10–20 while another may modify line 15. Our version-management system would detect the conflict and call out both sets of changes to me so that I could resolve the issue.

Before we installed our version-management system, the need to perform simultaneous development created the potential for another, very embarrassing, mistake: Suppose that we had received an order for both Prime and PC versions midway through the process described above, at a point when the PC version had been changed and debugged but the changes had not been merged into the Prime version. Under our old system, a misguided person with good intentions could very easily have selected the latest tested Prime and PC versions to build the product. The product probably would not have worked, because the two versions would have been incompatible.

Now, using the version-management system, once we have proven out a product version, whether for the PC or the Prime, we define it as the working version of the product. When the version-manager system is called on to generate a version, it will automatically select the working version, which, in our scenario, would be the earlier versions of the Prime and PC products. Version management thus ensures that we always ship the latest proven version of our product.

Managing Variations

Another problem that we faced prior to installing version management was the need to manage a wide variety of variations of our program. For example, our Multi-CAD product offers dozens of optional menuing systems that can make the user interface simulate the user's mainframe system of choice. Since the program features on-line help, each menuing system must be associated with its own slightly different help system. We have two documentation people who are responsible for developing the on-line help systems. During in-house testing, we frequently found that the programmers developing the menuing systems were using either the wrong help system or one that was outdated—the menuing system had been completed, but the disk hadn't been carried across the hall and put on the correct machine.

The version-management system, in combination with a local area network, easily resolved this problem. When we first began using version management, each programmer worked on a stand-alone PC. After using the version manager for a while, we realized that we could improve our operations even further by tying them all together on a development network. We use a Sun Microsystems host, with a large hard disk and a tape backup unit, as the file server. Now, the documentation programmers simply place the latest update on the network file server, and the version manager automatically associates it with the correct menuing systems.

Creating Makefiles

When I first got involved with the project, we were manually creating dependency files (also known as makefiles). In the process of simultaneously creating a mainframe version of the program, we had to keep track of any additional files used by the Prime version. This was not the easiest task: Since we used a UNIX-style make utility, in which each dependency statement is independent of the other dependency statements, it is almost impossible to determine the structure of the program by reading the dependency file.

This problem was solved by integrating our version-management system with make- and dependency-tracking utilities (also from Seidl), making it possible to automatically reconstruct any product version with a single command. With our dependency-tracking utility, updating a

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

dependency file to include information about files that we've added to our system became a simple task. The input data used by our dependency-tracking utility is called pre-DDL, which, in its simplest form, can be a list of filenames that do not even need to include suffixes. All we have to do is add the name of the new file used by the Prime version of the code to the pre-DDL file. The dependency tracker automatically creates a new dependency file, which can be used by the make utility to build the system.

We recently discovered an additional benefit of Seidl's dependency-tracking utility when we converted from Lahey FORTRAN and Lattice C to Microsoft compilers. This could have been a very time-consuming task had we still been generating dependency files by hand: We would have had to update each one manually, and it would have taken a fair amount of time to change the errors made through inability to read the cryptic dependency files. With the dependency tracker, all we had to do was change a few parameters, and all of our dependency files were automatically updated to reflect the move to Microsoft. Simultaneous programming took place with no downtime.

Making Backups

The version-management system has also relieved us of the headache of making frequent manual backups of our source files. Prior to the installation of this system, we had a disk crash that cost us dearly. After this occurred, we became very diligent about backing up files but found ourselves saddled with a tedious job, one that ate up a considerable amount of time that could have been better spent on productive programming.

The version-management system protects us from accidental or deliberate misuse, media failure, communication errors, power loss, and other disasters by providing automatic backups at any time interval that we specify. The version manager will archive only files that have been changed since the last archive unless we specify that *everything* is to be saved. This makes archiving a very simple task and ensures that it will be done often.

The backup methods employed are virtually "bombproof." First, a temporary file is created in the same directory as the original archive file. The primary portion of the temporary filename is the same as the original archive filename. (This ensures that the temporary file is on a nonvolatile medium.) Next, the original archive file is copied to a backup directory and then deleted. Finally, the temporary file is renamed to become the new archive file. Every record in archive files contains a CRC checksum, which is validated each time the file is accessed. If data appears to be corrupted, the user is notified so that the problem can be quickly corrected. Since moving to this system, we have not lost a single line of code.

Maintaining Revisions

Version management provides yet another important benefit: the ability to regenerate any version of our product. In the past, we spent a considerable amount of time saving and labeling each revision that was made because of the possibility that a future editing effort might turn out to be infeasible. Without this tedious effort, a potential disaster—reaching a blind alley and being unable to locate the previous revision—is always right around the corner. The version-management system maintains for each project a complete revision history of each source file, documentation file, dependency file, and so on. Any revision of any file may be extracted from the archives. This feature allows programmers to backtrack to any previous revision for any reason. Revisions may be extracted from the archives by any of several criteria (including revision number, version label, time stamp, or revision ID) and placed in a work directory.

The version-management system's revision-history feature comes in handy in other ways, too. For example, suppose I notice that a program is running significantly slower than it did before. I'd like to know why the change that slowed the program was made so that I can make an intelligent decision as to whether or not it was really necessary. If the programmer is around and remembers why he or she made the change, there's no problem—but suppose the programmer is on vacation, has left the company, or simply can't

The
Heap Expander™
version 2.05

Now your programs can have virtually unlimited heap space using expanded memory, extended memory, disk space, or any combination of the three. And it's all transparent. The Heap Expander's initialization code checks the system's resources and uses whatever is available.

only $79.95*

- Uses LIM-standard expanded memory if present.
- Uses AT-style extended memory if present.
- Swaps data to disk as needed.

Libraries and Source Code for:
- Turbo C
- Microsoft C 4.0 and 5.0
- Turbo Pascal 3.0 and 4.0

Requires an IBM PC, XT, AT, or close compatible with MS-DOS or PC-DOS version 2.0 or above

MC/VISA/COD call
1-800-248-1045 x 100 (US)
1-800-952-5560 x 100 (Idaho)

*CA residents add 6 1/2% sales tax.
Foreign customers, please add $4.00 for shipping and handling.

The Tool Makers
P. O. Box 2151
Santa Cruz, CA 95063
(408) 458-0690

Number 166 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

remember. This is why it's so important to me to have a record of who made the change, when it was made, why it was made, and so forth.

We used to accomplish this by producing text files on a word processor and saving them on the same disk that held our source files, but the version-management system maintains a very good audit trail for our project while taking almost no time away from our programming effort. Each time a revision is made, the version manager automatically records the time and date it was made, the user who placed it in the archive, and the revision from which it was derived. The version manager also prompts the user for a brief comment on why the change was made. All this information is stored with the revision and can easily be retrieved as an on-screen or printed report.

In conclusion, the version-management and associated product-generation utilities (which use Seidl's Version Manager and Make and SMKgen utilities) form a very efficient file clerk. Version management has eased our project-management chores and increased our code portability, productivity, and archiving security. It has also freed me to spend my time where it should be spent: managing. PJ

Ken Cunliffe has served as software manager with XYSYS for the past four years. Previously, he was technical director with CADSTEEL in Stockport, U.K. He studies electronic engineering at the Bolton Institute of Technology in the United Kingdom.

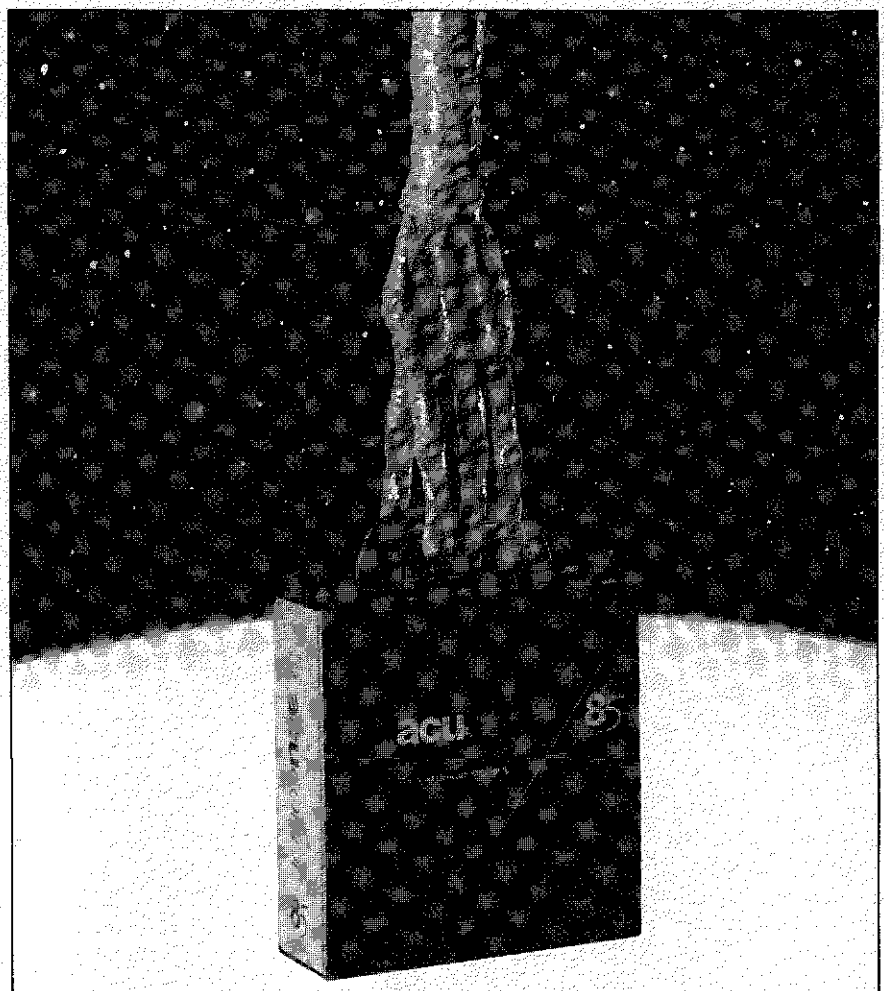
Version Management Vendors

TLIB
Burton Systems Software
P.O. Box 4156
Cary, NC 27519-4156
919-856-0475

MKS RCS
Mortice Kern Systems, Inc.
35 King Street N.
Waterloo, Ontario
N2J-2W9 Canada
519-884-2251

PVCS
Polytron Corporation
1700 NW 167th Pl.
Portland, OR 97006
503-645-1150

SVM and SMKgen
Seidl Computer Engineering
3106 Hilltop Dr.
Ann Arbor, MI 48103
313-662-8086



INTRODUCING
THE MOST PORTABLE COBOL
IN THE UNIVERSE

Go where you want to go
Transport COBOL programs to almost any platform—without changing one line of code. Run on AIX, MS-DOS, NETBIOS, UNIX, OS/2, ULTRIX, XENIX and VMS without recompiling. No other COBOL-85 offers such hardware independence, or the instantly-expanded possibilities.

Open new windows in your applications
Create better-looking programs with windows. Add help screens, pop-up/pull-down menus, context-sensitive messages, more. ACUCOBOL-85 has a Display Window verb, including extensions for boxed, reverse-video and pop-up windows in color. Paint special forms on the screen with our Display Line verb. And scroll up/down inside windows.

Don't get lost in space:
ACUCOBOL-85 keeps you on course
ACUCOBOL-85 is NBS certifiable error-free under the High 1974 and Intermediate 1985 ANSI standards. If you lose course, it has its own interactive source code debugger that's fast and easy to use.

Step into the future
Make your job simpler and stay competitive—even ahead of the game. Put the next generation COBOL to work for you today.

Free evaluation copy
Call (619) 271-7097 today for your free evaluation copy of ACUCOBOL-85. And see the shape of things to come.

acucobol inc

Voice: (619) 271-7097 Fax: (619) 566-3071
7950 Silverton Avenue Suite 201 San Diego California 92126

ACUCOBOL-85/the ACUCOBOL logo are trademarks of ACUCOBOL, Incorporated; MS-DOS/XENIX of Microsoft Corporation; NETBIOS/OS/2/AIX of International Business Machines Corporation; VMS/VAX/ULTRIX of Digital Equipment Corporation; UNIX of AT&T Bell Laboratories; RM/COBOL and RM/COBOL-85 of Ryan-McFarland Corporation.

Number 282 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Still-Higher 256-Color VGA Resolutions

by
Michael Abrash

Installment 14: In which we crank standard VGAs up to 360x480 256-color resolution, exceeding IBM's official 256-color resolution by a mere 170 percent, and also look at a way to pin down that ever-present bugbear of PC graphics programming—performance.

Several installments ago, we learned how to coax 320x400 256-color resolution out of a standard VGA. At the time, I noted that the VGA was actually capable of supporting 256-color resolutions as high as 360x480, but I didn't pursue the topic further, preferring to concentrate on the versatile and easy-to-set 320x400 256-color mode instead.

I got more feedback on that installment than on any other in a long time, illustrating once again that nothing fascinates graphics programmers like superior color–resolution combinations (especially undocumented ones). One particularly useful item came from John Bridges, who sent along a complete mode-set routine for 360x480 256-color mode, which he has placed into the public domain. John writes, "I also have a couple of freeware (free, but not public domain) utilities out there, including PICEM, which displays .PIC, .PCX, and .GIF images, not only in 360x480x256, but also in 640x350x256, 640x400x256, 640x480x256, and 800x600x256 on SuperVGAs. PICEM is in version 1.9b and can be found on all the above systems." (I've edited the quote slightly; John sent the information via MCI Mail, which tends to induce the dropping of certain parts of speech and punctuation.)

In this column, I'm going to combine that mode-set code with appropriately modified versions of the dot-plot code from the earlier 256-color installment and the line-drawing code from two installments back. Together, those routines will make a pretty nifty demo of the capabilities of 360x480 256-color mode.

I'll also look at Paradigm's Inside! Turbo C performance-analysis software. While that's not directly related to graphics, on the PC—with its limited processor power and bizarre graphics adapters—assembler, performance, and good graphics go hand in hand. Timing software like Inside! Turbo C, used regularly, can teach you more about creating good graphics programs than any number of books and articles. Why? Because every graphics program has unique performance characteristics, and only by measuring can you determine true performance.

In sum, we'll be exploring both higher 256-color resolution and ways to improve performance. That's a neat package, for we'll be approaching graphics from both ends: better appearance and better performance. Let's get started!

Extended 256-Color Modes

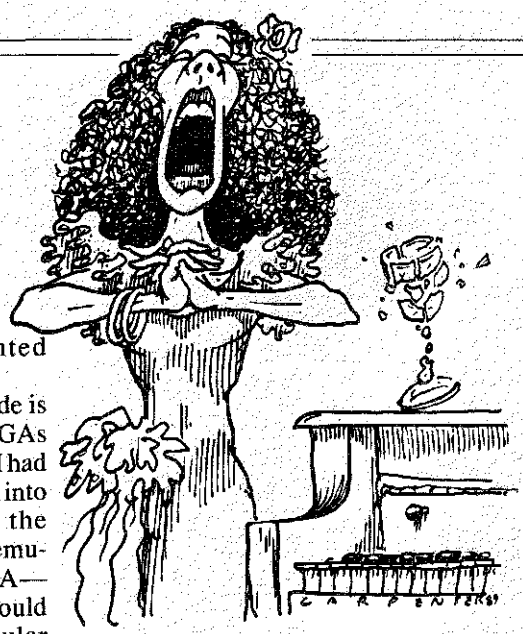
When last we left 256-color programming, we had found that the standard 256-color mode, mode 13H, which officially offers 320x200 resolution, actually displays 400, not 200, scan lines, with line doubling used to reduce the effective resolution to 320x200. By tweaking a few of the VGA's mode registers, we converted mode 13H to a true 320x400 256-color mode. As an added bonus, that 320x400 mode supports two graphics pages—a distinct improvement over the single graphics page supported by mode 13H. (However, we also learned how to get *four* graphics pages at 320x200 resolution, should that be needed.)

I particularly like 320x400 256-color mode for two reasons: It supports two-page graphics, which is very important for animation applications, and it doesn't require changing any of the monitor-timing characteristics of the VGA. The mode bits that we changed to produce 320x400 256-color mode are pretty much guaranteed to be the same from one VGA to another, but the monitor-oriented registers are less certain to be constant, especially for VGAs that provide special support for the extended capabilities of various multiscanning monitors.

All in all, those are good arguments for 320x400 256-color mode. However, the counter-argument seems compelling as well: Nothing beats higher resolution for producing striking graphics. Given that, and given that John Bridges was kind enough to make his mode-set code available, I'm going to look at 360x480 256-color mode next. However, bear in mind that the drawbacks of this mode are the flip side of the strengths of 320x400 256-color mode: only one graphics page, and direct setting of the monitor-oriented registers. Also, this mode has a peculiar and unique aspect ratio, with 480 pixels vertically (as many as high-resolution mode 12H) and only 360 horizontally. That makes for fairly poor horizontal resolution and sometimes-jagged drawing. On the other hand, the resolution is better in both directions than in mode 13H, and mode 13H also has an odd aspect ratio; so, it seems a bit petty to complain.

The single graphics page isn't a drawback if you don't need page flipping, of course, and so there's not much to worry about there: If you need page flipping, don't use this mode. The direct

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



setting of the monitor-oriented registers is another matter.

I don't know how likely this code is to produce problems with clone VGAs in general; however, I did find that I had to put a Video Seven VRAM VGA into "pure" mode—where it treats the VRAMs as DRAMS and exactly emulates a plain-vanilla IBM VGA—before 360x480 256-color mode would work properly. Now, that particular problem was due to an inherent characteristic of VRAMs and shouldn't occur on Video Seven's Fastwrite adapter or any other VGA clone. Nonetheless, 360x480 256-color mode is a good deal different from any standard VGA mode, and while the code in this article runs perfectly well (in my experience), I can't guarantee its use on any particular VGA–monitor combination, unlike 320x400 256-color mode. Mind you, 360x480 256-color mode *should* work on all VGAs—there are just too may variables involved for me to be certain. Feedback from readers with broad 360x480 256-color experience is welcome.

The above notwithstanding, 360x480 256-color mode offers 64 times as many colors and nearly 3 times as many pixels as the original Color/Graphics Adapter color-graphics mode, making startlingly realistic effects possible. No mode of the VGA, documented or undocumented (at least none that I know of), offers a better combination of resolution and color—even 320x400 256-color mode has 26 percent fewer pixels. In other words, 360x480 256-color mode is worth considering. Let's have a look.

360x480 256-Color Mode

I'm going to start by showing you 360x480 256-color mode in action, after which we'll look at how it works. I suspect that once you see what this mode looks like, you'll be more than eager to learn how to use it.

Listing 1 contains three C-callable assembly language functions. As you'd expect, **Set360x480Mode** places the VGA into 360x480 256-color mode. **Draw360x480Dot** draws a pixel of the specified color at the specified location. Finally, **Read360x480Dot** returns the color of the pixel at the specified location. (This last function isn't used in the example program in this article, but it is included for completeness.)

Listing 2 contains a modification of the Turbo C line-drawing code I discussed back in installment 12 (*PJ* 7.3). The code has been altered to select 360x480 256-color mode and to cycle through all 256 colors that this mode supports, drawing each line in a different color.

The first thing you'll notice when you run this code is that the speed of 360x480 256-color mode is pretty good, especially considering that most of the program is implemented in C. In general, drawing in 360x480 256-color mode can sometimes actually be *faster* than in the 16-color modes, because the byte-per-pixel display-memory organization of 256-color mode eliminates the need to read display memory before writing to it to isolate individual pixels.

The second thing you'll notice is that exquisite shading effects are possible in 360x480 256-color mode; cone-shaped sections of lines blend into one another remarkably smoothly, even with the default palette. The VGA allows you to select your 256 colors from a palette of 256,000, so you could, if you wished, set up the colors to produce less detail but still-finer shading. (We'll get to reprogramming the palette someday—there's so much to do with the VGA!)

The one thing you may not notice right away is just how much detail is visible on the screen, because the blending of colors tends to obscure the superior resolution of this mode. Each of the four rectangles displayed measures 180 pixels horizontally by 240 vertically. Put another way, each *one* of those rectangles has two-thirds as many pixels as the entire mode-13H screen—in all, 360x480 256-color mode has 2.7 times as many pixels as mode 13H! As mentioned above, the resolution is unevenly distributed, with vertical resolution matching that of mode 12H but horizontal resolution barely exceeding that of mode 13H. But resolution is hot stuff, no matter how it's laid out, and 360x480 256-color mode has the highest 256-color resolution you're likely to see on a standard VGA.

Now that we've seen the wonders of which our new mode is capable, let's take the time to understand how it works.

How This 256-Color Mode Works

In discussing 360x480 256-color mode, I'm going to assume that you're familiar with my discussion of 320x400 256-color mode that appeared in *PJ* 7.1; if not, get hold of that issue (January/February 1989) and read it. The two modes have a great deal in common, but I don't want to bore faithful "On Graphics" readers and use up this always-too-small space by repeating myself.

360x480 256-color mode is essentially 320x400 256-color mode stretched in both dimensions. Let's look at the vertical stretching first, since that's the simpler of the two.

480 Scan Lines: A Little Slower

There's nothing particularly unusual about 480 scan lines—standard modes 11H and 12H support that vertical resolution. The number of scan lines has nothing to do with either the number of colors or the horizontal resolution; so, converting 320x400 256-color mode to 320x480 256-color mode is a simple matter of reprogramming the VGA's vertical control registers—which control the scan lines displayed, the vertical sync pulse, vertical blank-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

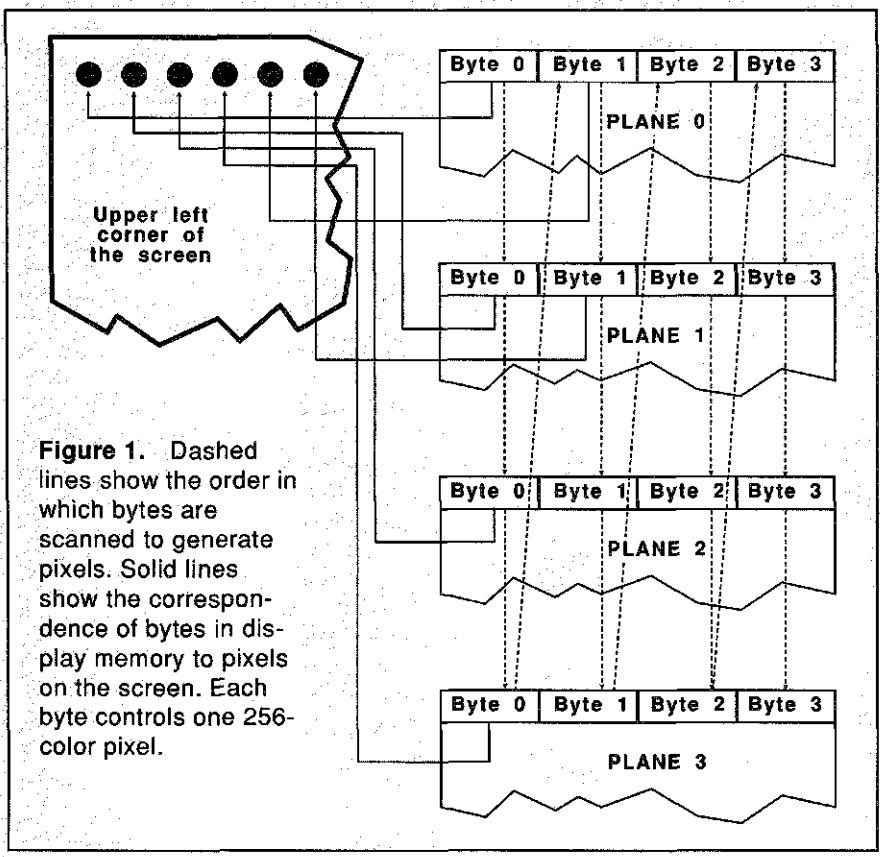


Figure 1. Dashed lines show the order in which bytes are scanned to generate pixels. Solid lines show the correspondence of bytes in display memory to pixels on the screen. Each byte controls one 256-color pixel.

ing, and the total number of scan lines—to the 480-scan-line settings and setting the polarities of the horizontal and vertical sync pulses to tell the monitor to adjust to a 480-line screen.

Switching to 480 scan lines has the effect of slowing the screen-refresh rate. The VGA always displays at 70 Hz, *except* in 480-scan-line modes; there, owing to the time required to scan the extra lines, the refresh rate slows to 60 Hz. (VGA monitors always scan at the same rate horizontally; that is, the distance across the screen covered by the electron beam in a given period of time is the same in all modes. Consequently, adding extra lines per frame requires extra time.) A rate of 60 Hz isn't *bad*—that's the only refresh rate the EGA supports—but it does tend to flicker a little more and so is a little harder on the eyes than 70 Hz.

360 Pixels per Line: No Mean Feat

Converting from 320 to 360 pixels per scan line is more difficult than converting from 400 to 480 scan lines per screen. None of the VGA's graphics modes supports 360 pixels across the screen—or anything like it; the standard choices are 320 and 640 pixels across. However, the VGA *does* support the horizontal resolution we seek, 360 pixels, in 40-column text mode.

Unfortunately, the register settings that select those horizontal resolutions aren't directly transferable to graphics mode. Text modes display nine dots (the width of one character) for each time information is fetched from display memory, while graphics modes display just eight dots per display-memory fetch. (Although it's a bit confusing, it's standard terminology to refer to the interval required for one display-memory fetch as a "character," and I'll follow that terminology from now on.) Consequently, both modes display either 40 or 80 characters per scan line; the only difference is that text modes display more pixels per character. Given that graphics modes *can't* display nine dots per character (there's enough information only for eight 16-color pixels or four 256-color pixels in each memory fetch, and that's that), we'd seem to be at an impasse.

The key to solving this problem lies in recalling that the VGA is designed to drive a monitor that sweeps the electron beam across the screen at exactly the same speed, no matter what mode the VGA is in. If the monitor always sweeps at the same speed, how does the VGA manage to display both 640 pixels across the screen (in high-resolution graphics modes) and 720 pixels across the screen (in 80-column text modes)? Good question. The answer is that the VGA has, not one, but *two* clocks on board, and one of those clocks is just enough faster than the other clock so that an extra 80 (or 40) pixels can be displayed on each scan line.

In other words, there's a slow clock (about 25 MHz) that's usually used in graphics modes to get 640 (or 320) pixels on the screen during each scan line, and a second, faster clock (about 28 MHz) that's usually used in text modes to crank out 720 (or 360) pixels per scan line. In particular, 320x400 256-color mode uses the 25-MHz clock.

You can probably see where I'm headed: We can switch from the 25-MHz clock to the 28-MHz clock in 320x480 256-color mode in order to get more pixels. It takes two clocks to produce one 256-color pixel, so we'll get 40 rather than 80 extra pixels by doing this, bringing our horizontal resolution to the desired 360 pixels.

Switching horizontal resolutions sounds easy, doesn't it? It's not. There's no standard VGA mode that uses the 28-MHz clock to draw eight rather than nine dots per character, and so the timing parameters have to be calculated from scratch. John Bridges has already done that for us, but I want you to appreciate that producing this mode took some work. The registers controlling the total number of characters per scan line, the number of characters displayed, the horizontal sync pulse, horizontal blanking, the offset from the start of one line to the start of the next, and the clock speed all have to be altered to set up 360x480 256-color mode. The function **Set360x480Mode** in Listing 1 does all that, as well as setting up the registers that control vertical resolution.

Once that's done, the VGA is in 360x480 mode, awaiting our every high-resolution, 256-color graphics whim.

Accessing Display Memory

Setting up for 360x480 256-color mode proved to be quite a task. Is drawing in this mode going to be as difficult?

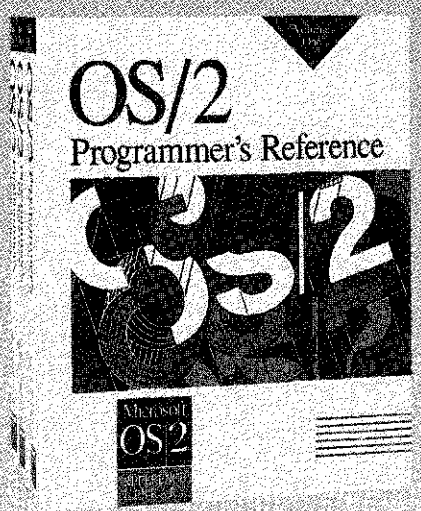
No. In fact, if you know how to draw in 320x400 256-color mode, you also know how to draw in 360x480 256-color mode; the conversion between the two is a simple matter of changing the working screen width from 320 pixels to 360 pixels. In fact, if you were to take the 320x400 256-color pixel-reading and -writing code from installment 10

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Now... An Affordable Path to Presentation Manager Software Development

You no longer need to make a significant financial investment to develop Microsoft OS/2 Presentation Manager applications. Here's a low-cost solution to help you explore the Application Programming Interface (API) and start creating intuitive, easy-to-use software applications for the graphical environment of Presentation Manager.

Whether you're just testing the Presentation Manager development waters or are ready for serious programming, the Microsoft OS/2 Programmer's Reference Library will be your cornerstone resource. These volumes — companions to Microsoft's Softset package of OS/2 Presentation Manager software development tools — are packed with detailed information on every MS OS/2 system function, related data types, macros, structures, messages, and file formats.



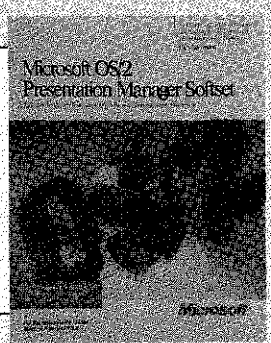
Volume 1. A thorough overview of the MS OS/2 system functions and the concepts and principles behind the functions. Here's the fundamental information you need for an understanding of MS OS/2. $29.95

Volume 2. A comprehensive alphabetic reference of all the Presentation Manager functions that address the window management and graphics features of MS OS/2. Complete with information on syntax; descriptions of each function's action and purpose; parameters and field definitions; return values, error values, and restrictions; source-code examples; and programming notes. $29.95

Volume 3. Similar in format to Volume 2, this volume covers all the MS OS/2 base system functions that use the operating system to carry out tasks such as reading from and writing to disk files; allocating memory; starting other programs; and controlling the keyboard, mouse, and video screen. $19.95

A Complete, Low-Cost Software Development Package

When you combine the Microsoft OS/2 Programmer's Reference Library with the Microsoft OS/2 Presentation Manager Softset and a Microsoft language that supports OS/2 you'll have the complete solution for developing intuitive, graphical applications under OS/2 Presentation Manager. The Softset — a package of affordably-priced, fully-documented visual software tools that include dialog, icon, and font editors, a resource compiler, and other tools — and the Microsoft OS/2 languages are available at your languages specialty dealer.



ORDER TODAY

The Microsoft OS/2 Programmers Reference Library is available wherever books and software are sold or you can order directly from Microsoft Press.

YES, please send me these Microsoft OS/2 Programmer's Reference Library volumes:

_____ copies of Volume 1 at $29.95 ..... _____
[Order Code 86-97302]

_____ copies of Volume 2 at $29.95 ..... _____
[Order Code 86-97310]

_____ copies of Volume 3 at $19.95 ..... _____
[Order Code 86-97328]

_____ sets of all three of the above books. SAVE! Only $79.85. We'll pay postage the $4.00 postage ..... _____

Add sales tax for your state ..... _____

Postage and Handling:
$2.50 for one book; $.75 for each additional book.
We pay postage if you order all 3 volumes ..... _____

TOTAL ..... _____

PAYMENT: ☐ Check/Money Order ☐ MasterCard ☐ American Express ☐ VISA

CREDIT CARD NO. _____ EXP. DATE _____

NAME _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

Send your order to:
Microsoft Press, P. O. Box 1532, Hagerstown, MD 21742
Or call **800-638-3030** (8AM to 4PM EST, M-F). Refer to ad PJ889.

Microsoft® Press

Computer Books That Make Sense

PJ889

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

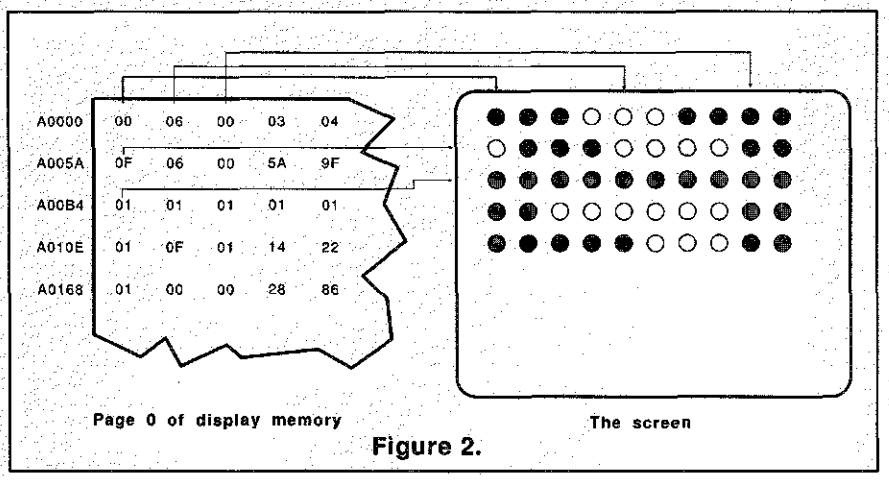


Figure 2.

(*PJ* 7.1) and change the **SCREEN_WIDTH** equate from 320 to 360, those routines would work perfectly in 360x480 256-color mode. (The program as a whole would also have to be changed to use a 360x480 256-color mode set before it would work properly.)

The organization of display memory in 360x480 256-color mode is almost exactly the same as in 320x400 256-color mode (which was covered in detail in installment 10). Each byte of display memory controls one 256-color pixel, just as in mode 13H. The VGA is reprogrammed by the mode set so that adjacent pixels lie in adjacent planes of display memory. Figure 1 (reprinted from installment 10) shows the organization of the first few pixels on the screen; the bytes controlling those pixels run cross-plane, advancing to the next address only every fourth pixel. The address of the pixel at screen coordinate (x,y) is

$$address = [(y * 90) + x] / 4$$

and the plane of a given pixel is

$$plane = x \text{ modulo } 4$$

(Thanks to a reader from Chips & Technologies for pointing out that in installment 10, I mistakenly exchanged x and y in the address and plane equations for 320x400 256-color mode. As he diplomatically pointed out, the equations I gave won't work at all. The correct equations for 320x400 256-color mode are

$$address = [(y * 80) + x] / 4$$
$$plane = x \text{ modulo } 4$$

Fortunately, the code in that installment does implement the correct calculations. I'd gladly thank by name the gentleman from C & T who provided this correction; alas, I've lost his letter. Whoever you are, drop me a line and I'll be most pleased to credit you.)

A new scan line starts every 360 pixels, or 90 bytes, as shown in Figure 2. This is the major programming difference between the 360x480 and 320x400 256-color modes; in the 320x400 mode, a new scan line starts every 80 bytes.

The other programming difference between the two modes is that the area of display memory mapped to the screen is longer in 360x480 256-color mode—which is only common sense, given that there are more pixels in that mode. The exact amount of memory required in 360x480 256-color mode is 360 times 480, or 172,800 bytes. The more perceptive among you will note that that's more than half of the VGA's 256Kb memory complement, which means that page-flipping is out. However, there's no reason you couldn't use that extra memory to create virtual screens larger than 360x480, which you could then scroll around, if you wish. (Horizontal smooth panning is probably not possible, owing to the unusual pixel formatting of this mode. I haven't tried it, however; if any of you out there have tried 360x480 256-color smooth panning, please let me know how it worked.)

That's really all there is to drawing in 360x480 256-color mode. From a programming perspective, this mode is no more complicated than 320x400 256-color mode once the mode set is completed, and it should be capable of good performance given some clever coding. It's not particularly easy to implement bitblt, block-move, or fast line-drawing code for either of the extended 256-color modes, but it can be done—and it's worth the trouble. Even

GRAPHICS!

Add lightning fast graphics to your programs quickly and easily through the popular PCX file format. Why reinvent the wheel? Make your programs immediately compatible with hundreds of packages from Aldus PageMaker to ZSoft's PC Paintbrush with these linkable graphic libraries.

"A Professional Class Product" - DBAdvisor, Jan '89

NEW! Version 3.5 of the **PCX Programmer's Toolkit** gives you over 60 powerful functions to manipulate bitmapped graphics. Use Virtual screens, Super VGA modes, LIM 4.0 support, a 300 page manual, 9 utilities including screen capture and display, and the fastest routines on the market. **$195**

Need Special Effects, but caught in a GRASP?

Why create a demo when you can create the real thing? Don't be trapped in a slideshow editor or demo program when you can use **PCX Effects** and your favorite programming language. A complete Music Language and spectacular special effects for exploding your graphics! **$99**

Blazing Graphics Text

With **PCX Text** you can display text with graphics as fast as it always should have been. Display characters, strings, input fields, font scaling, background transparency, and more. Includes a font editor, fonts, and conversion utility for blazing graphics bitmapped text. **$99**

All packages support 12 compilers for C, Pascal, Basic, Fortran, Assembly, and Clipper. All modes of the Hercules, CGA, EGA, VGA, and Super VGA adapters are supported, up through 800x600x256 (22 modes in all). Assembly Language source code is optionally available. Trademarks are property of their respective holders.

No Royalties! 30-day Money Back Guarantee.
VISA/MC/AMEX/COD/PO accepted.

G·E·N·U·S MICROPROGRAMMING
11315 Meadow Lake • Houston, Texas 77077 • (713) 870-0737

1-800-227-0918

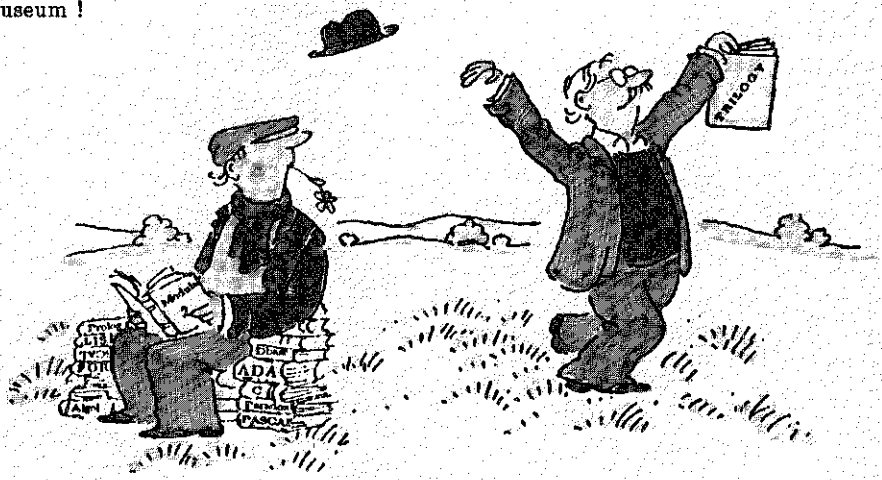
Number 261 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HAVE YOU REALIZED THAT

YOUR LANGUAGE IS A DINOSAUR !!!

THE ONE-PARADIGM LANGUAGE YOU ARE USING NOW, such as a third generation procedural language (Pascal, Modula, C, ADA, etc.), a fourth generation database language (Dbase, SQL, etc.) or a fifth generation logic language (Prolog, Lisp), **IS FACING EXTINCTION !** Put it in a museum !



TODAY, programmers around the world use TRILOGY - the language designed to *'create a revolution'* (R.Fischer, PC Welt, Germany)! **TOMORROW you will be one of them, and you will love it!**

TRILOGY introduces the new Multi-Paradigm Generation of computer languages! It seamlessly integrates the last three generations of languages into a single genuine **GENERAL PURPOSE LANGUAGE of unprecedented power** and expressivity, radiating simplicity and elegance. With this *'weapon that dramatically improves your productivity'* (Dr.J.Pournelle, BYTE), you can save 35-40 % of code.

Large-scale integration of software, until now believed to be unmanagable, was made possible by discovering the logic theory applicable to all programming paradigms. We have based **TRILOGY** on this theory, and we have managed to hide all of the difficult concepts behind a **friendly Pascal-like syntax** as easy as "if-then-else".

TRILOGY, an original product of inventive thinking, is the result of a **decade of research** and three years of implementation. We are offering you a *'deluxe experimenter's system'* (Ph. Chapnick, AI Expert) rejecting the unscientific *ad hoc* methods of language design used in all of the one-paradigm languages.

LOOK AT TRILOGY TAKING THE WORLD BY STORM :

'... Trilogy is what Pascal would be if it were designed today with the knowledge of logic and database programming we now have ...' Ray Weiss, Electronic Engin. Times, Dec. 1987, 'Pascal Joins Prolog'

'... The West Coast Computer Fair has long been the spawning ground for important micro products ... It does not happen so often now, but it's still possible, as for example Trilogy ... what I've been able to do with (Trilogy) really impressed me ... you may find yourself with a secret weapon that dramatically improves your productivity ... I am not alone in my impression that Trilogy may ... be one of the "big" languages like Pascal or Prolog.' Dr.Jerry Pournelle, Columnist, BYTE, July's 1988 column 'Computing At Chaos Manor'.

'... It's an incredibly fast language ... Everything I saw in this language told me that its author was no lightweight. And it was obvious, even in the manual, that this had been a long term project... (Trilogy is) too good a language and too good a development environment.' Dave Thompson, Editor-In-Chief, Micro Cornucopia, January 1988.

'... programmers will soon come to expect languages that support several paradigms. One such language already available, Trilogy, can be used for procedural, functional, logical, constraint, and data base programming. ... Prolog programs, using backtracking, typically require several hours to solve this problem (Zebra). Trilogy, using more powerful (built-in constraint satisfaction) decision procedures, finds the answer in less than one second.' Dr. Peter Gregory, the author of 'Programming in Pascal', Computer Language, April 1988.

'... Trilogy embodies a wealth of features that range from the unusual to the unheard of ... we expect to see Prolog-Trilogy religious wars break out ... If we had to enlist, we'd sign up with Trilogy.' Mickey Williamson, the author of 'Artificial Intelligence for Microcomputers', PC Week, February 9, 1988.

'... Trilogy is a new programming language which integrates the knowledge acquired in computing science in the last decade. The result is astonishing. In the competition for the most economic programming system, (Trilogy) ... has a good chance to win the first place ... When Trilogy gets the attention it deserves, it will create a revolution ... Ronald Fischer, PC WELT, Germany, November 1988, 'Pascal und Prolog in Einem'.

'... Trilogy is a unique product. It is the only product I have seen that combines logic programming with procedural and database concepts.' Alex Lane, Contributing Editor, BYTE, March 1988.

'... Trilogy ... is a deluxe experimenter's system.' Philip Chapnick, AI Expert, May 1988, 'Don't Avoid Conventions - Be Multilingual'.

'... The three levels of Trilogy offer the strength of conventional procedural programming languages, database languages, and declarative languages used in artificial intelligence ... (Trilogy) is much more simple than expected ... people with experience in programming ... will be able to learn it very fast.' Computerworld Switzerland, March 14, 1988, 'Die Trilogie der Sprachen'.

'... Trilogy, the new programming language from Canada that is receiving attention in the U.S. AI community, is adding an MS-DOS module and a Graphics capability (module) ... (Trilogy) is modular and includes a full screen editor, module librarian, interactive compiler, on-line linker and loader, cross-referenced help, and automatic make. Windowing and external file access are supported, as are 80x87 coprocessors. The Pascal-like syntax (fits into one page) ...' AI Expert, October 1988, 'Trilogy keeps growing'.

TRILOGY runs on IBM PC, XT, AT, and compatibles, under DOS.
Order from:
Complete Logic Systems
741 Blueridge Ave.
North Vancouver
B.C., V7R 2J5, Canada.
Telephone: (604) 986-3234

VISA, MC, money order welcome. Add US $ 7 for shipp./handl. in North America, or US $ 13 for oversees shipments.

TRILOGY ADVANCED PACKAGE: US $200.00
(package includes: TRILOGY V1.3, TOOLBOX, LINKER)
or you can order separately:
TRILOGY V 1.3: US $100.00 (Compiler & Environment)
TRILOGY TOOLBOX: US $ 50.00 (DOS connect., Graphics)
TRILOGY LINKER: US $ 50.00 (Creating .EXE files)
2 Demo disks: US $ 10.00 (Running Mini-Trilogy)

70% discount offered to all registered users of TRILOGY V1.3 on **TRILOGY++** with OOP capabilities to be released in 1990.

Number **301** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the small taste we've gotten of the capabilities of these modes shows that they put the traditional CGA, EGA, and (in many cases) even VGA modes to shame.

Measuring Performance

Graphics and performance go hand in hand on the PC. Given the relatively low performance of both the PC and standard PC graphics hardware, high-performance graphics code is a must. In the last installment, I presented a timer for analyzing code performance in detail at a relatively low (close to the hardware) level. That timer is excellent if you want to try to figure out if it's faster to increment a register twice or add 2 to the register, but more-sophisticated tools are needed to identify time-critical code in the context of whole programs and to analyze and improve the performance of algorithms and functions.

Inside! Turbo C (the exclamation point is part of the name), from Paradigm Systems, Inc. (P.O. Box 152, Milford, MA 01757, 508-478-0499, $125.00), is one such performance-analysis tool—and it's a good one. Inside! automatically uses the debugging information Turbo C generates to produce a function-by-function performance profile, which quickly reveals which functions are candidates for optimization. You can then zero in on the truly time-critical code by having Inside! analyze performance at a source-line level or by requesting a performance analysis of one or more specific events.

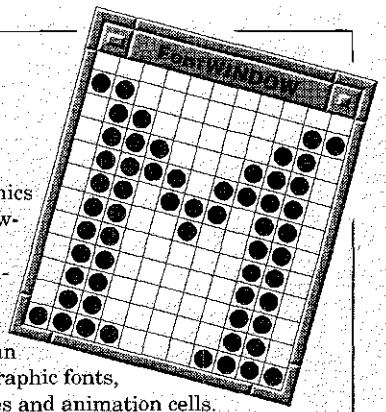
The best thing about all this is that it doesn't require any particular effort. You don't much need to think about what part of your program to optimize—all you need to do is read over the performance profiles Inside! generates. That, in turn, saves an enormous amount of guesswork, experimentation, and wasted optimization. Basically, Inside! makes it easy to ask the question "Where is my program spending its time?" Once you start asking that question, it's astonishing how often you'll find that it's possible to eliminate huge chunks of execution time.

Inside! is oriented toward use with C rather than assembly language programs. You can analyze assembly language code with Inside! (although a bug in TASM 1.0, which should by fixed in TASM 1.1, makes MASM a better choice right now), but that's more easily done with assembly language code that's part of a C program than with stand-alone assembly language. I *was* able to analyze stand-alone assembly language, but the manual doesn't provide much information, and it took me a good bit of fiddling around to get things working. (By the way, Inside!'s results matched those from the timer I presented in the last installment; however, because Inside! leaves interrupts enabled, it is both somewhat less accurate for short timing intervals and capable of timing much longer intervals.) Still, few substantial programs are written entirely in assembly language (who has the time to waste writing initialization code and the like in assembly language?), and those sizable C–assembly language hybrid programs are exactly the sort of programs for which the ability of Inside! to produce easily interpreted performance statistics is most useful.

Inside! is easy to use and reliable. Paradigm has encountered and dealt with some hairy performance-analysis problems that I wouldn't have cared to figure out for myself, such as running

Power Graphics for your PC!

Write powerful interactive graphic applications faster and easier using MetaWINDOW - the award winning leader in graphic development systems. MetaWINDOW, independently benchmarked the fastest graphics toolkit available, allows you to build tight custom, graphics applications without the massive bulk and "me too" look of other graphic systems.



The most powerful PC graphics toolkit now has the most powerful font editing system! FontWINDOW makes it simple for you to create, edit & customize your own fonts. Using FontWINDOW you can create virtually unlimited graphic fonts, icons, foreign language styles and animation cells.

Over 200 Power-Packed Functions

- State of the art event driven operation.
- Ultra-fast bitmap and outline text drawing functions.
- Extensive rasterOp flexibility makes it easy to rubber-band or drag complex objects.
- QuickDraw superset speeds porting PC and Mac programs.
- Dynamic runtime support for 100+ popular graphics cards.
- Advanced versions available to support the full range of super-VGA and ultra-quick TI 34010 coprocessor cards.
- Full, illustrated documentation and free technical support.
- NO royalties!

- More than 200 high quality fonts, icons, and foreign language styles - over 2 megabytes of fonts with NO royalties!
- Quickly merge characters from multiple fonts to create totally new custom designs.
- ZoomFont function makes it easy to scale fonts to virtually any size. Includes ZoomFont C source code for scaling fonts within your own programs.
- Full font format specifications, illustrated documentation and free technical support.

MetaWINDOW (supports 15 popular C & Pascal compilers) . . . $250
(single language for Turbo C, Turbo Pascal or MS Quick C) . . $125
FontWINDOW . . . . *Introductory price thru Oct. 1. . $95!* . . . . . ~~$125~~

30 Day No Risk Money-Back Guarantee!

Order Hotline 800-332-1550

Free MetaWINDOW & FontWINDOW Working Demos
Download direct 408-438-5368 (1200/2400 baud, X-modem/Kermit), or call 408-438-1550 for free demo disk and complete capabilities brochure.

METAGRAPHICS
SOFTWARE CORPORATION
269 Mount Hermon Road, Scotts Valley, CA 95066
Phone: 408-438-1550 FAX: 408-438-5379

Number 143 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

in virtual-8086 environments, which intercept interrupts in sometimes unpredictable ways. Inside! is not quite perfect, however. For one thing, the manual (like most) could use some work: There's no index, and the examples could be explained in considerably more detail. On the other hand, there's a tremendous amount of useful technical information packed into less than 100 pages, if you're willing to work a bit to understand it.

The only other problem I encountered with Inside! is that it doesn't handle very large programs. Understand that I mean *very* large programs: The program I couldn't analyze with Inside! requires a minimum of 540Kb free to run. I don't know of anything else that can analyze this program, either, so that probably isn't a fair criticism. That program is one I co-wrote, however, and I very much wanted to use Inside! on it, but the two packages just couldn't fit into memory together. At any rate, the people at Paradigm have workarounds for analyzing all but the most severely memory-constrained programs and are working on using overlays, expanded memory, and the like to eliminate the problem in the next release.

All in all, Inside! Turbo C is a must-have tool for C programmers concerned with performance. That's particularly true because Inside! makes it so easy for programmers to raise their optimization sights from individual statements and instructions to the algorithm–function level, where the greatest potential savings often await, and to identify efficiently code that's worth optimizing.

Versions of Inside! for Microsoft C, Turbo Pascal, and other compilers are also available.

In Conclusion

For now, we've come to the end of the 256-color rainbow. So far as I know, there aren't any more 256-color modes hiding in the standard IBM VGA. Of course, there are several *SuperVGA* 256-color modes around, and the 8514/A—with a 1024x768 256-color mode—is coming on strong. (Look for hardware-compatible clones of the 8514/A soon; when those become widely available and prices drop, the move to the next PC-graphics standard may well begin in earnest.) If there's enough reader interest in either SuperVGA modes or the 8514/A, I'll cover them; for now, though, I'll keep on keeping on with the VGA. Next time, I'll look into the potent combination of the VGA's split-screen and panning features, and in the not-too-distant future, I'll move on to drawing circles, ellipses, and the like . . . and drawing them fast, as always.

See you next time. PJ

Michael Abrash is a senior software engineer for Orion Instruments, Inc., in Redwood City, California, and a contributing editor of Programmer's Journal. Power Graphics Programming, *an anthology of Michael's "On Graphics" articles published by Que Corporation and* PJ, *is now available in bookstores and by mail order. Michael is also the author of* The Zen of Assembler, *due out this fall from Scott, Foresman & Company. Readers may write to Michael at 1599 Bittern Drive, Sunnyvale, CA 94087, or contact him via MCI Mail, username* MABRASH.

Code follows.

PRINTED GRAPHICS IN HIGH RESOLUTION!

GRAPHLINK™

THE PRINTER GRAPHICS SOLUTION™

GraphLink is the Printer Graphics Toolkit that does for your printer what other graphics packages do for your screen -- at the printer's resolution! *GraphLink is the solution you've been waiting for because GraphLink . . .*

- Does *NOT* do a low resolution screen dump -- it builds its own image at the *printer's* resolution - *even at 300 dots-per-inch for a LaserJet!*
- Provides more than 70 routines which closely emulate Borland's BGI graphics commands, including line, bar, ellipse, arc, pie slice, area fill, text, and viewport commands.
- Works with the most popular printers - including the HP LaserJet and DeskJet, and Epson, IBM, Toshiba, NEC, and other 9 and 24 pin dot matrix printers.
- Supports Turbo C, Microsoft C, Turbo Pascal, Quick Pascal, TopSpeed Modula-2, and other popular compilers.

Includes an excellent manual, free technical support, and a 30 day money back guarantee, all for just $125 (or $225 with source code)! (Foreign/Canada orders please add $10/$5.) *Order today, and solve your printer graphics problems!*

Visitech Software (412) 733-4775

05 3807 Ridgewood Ct., Pittsburgh, PA 15239 USA

Number 304 on the Reader Service Card

34010 Video Co-Processors from PC Tech

- Mono 2048 x 1536 displayed pixels
- Color 1024 x 768 displayed pixels
- Up to 5 M byte memory
- Optional CCITT imaging, TIGA, & DGIS software drivers
- C Compiler, Assembler, Linker, Debugger, DGIS Development Kit & TIGA Toolkit available
- PC/XT/AT compatible
- Custom designs available
- **PC Tech understands the 34010**

For further information, please contact:

PC Tech

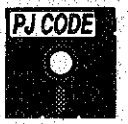
907 N. 6th St.
Lake City, MN 55041
(612) 345-4555 (Voice)
(612) 345-5514 (FAX)

PC Tech is a trademark of PC Tech, Inc.

PC, XT, AT, DGIS, and TIGA are trademarks or registered trademarks of their respective companies.

Number 208 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800-234-0386.
Mail-order information on pages 8 and 88.

Code from "Still-Higher . . . Resolutions," by Michael Abrash

1

LISTING 1

```
;
; Turbo C tiny/small/medium model-callable assembler
; subroutines to:
;       * Set 360x480 256-color VGA mode
;       * Draw a dot in 360x480 256-color VGA mode
;       * Read the color of a dot in 360x480 256-color VGA mode
;
; Assembled with TASM 1.0.
;
; The 360x480 256-color mode set code and parameters were provided
; by John Bridges, who has placed them into the public domain.
;
VGA_SEGMENT     equ     0a000h  ;display memory segment
SC_INDEX        equ     3c4h    ;Sequence Controller Index register
GC_INDEX        equ     3ceh    ;Graphics Controller Index register
MAP_MASK        equ     2       ;Map Mask register index in SC
READ_MAP        equ     4       ;Read Map register index in GC
SCREEN_WIDTH    equ     360     ;# of pixels across screen
WORD_OUTS_OK    equ     1       ;set to 0 to assemble for
                                ; computers that can't handle
                                ; word outs to indexed VGA registers
;
_DATA   segment public byte 'DATA'
;
; 360x480 256-color mode CRT Controller register settings.
; (Courtesy of John Bridges.)
;
vptbl   dw      06b00h  ; horz total
        dw      05901h  ; horz displayed
        dw      05a02h  ; start horz blanking
        dw      08e03h  ; end horz blanking
        dw      05e04h  ; start h sync
        dw      08a05h  ; end h sync
        dw      00d06h  ; vertical total
        dw      03e07h  ; overflow
        dw      04009h  ; cell height
        dw      0ea10h  ; v sync start
        dw      0ac11h  ; v sync end and protect cr0-cr7
        dw      0df12h  ; vertical displayed
        dw      02d13h  ; offset
        dw      00014h  ; turn off dword mode
        dw      0e715h  ; v blank start
        dw      00616h  ; v blank end
        dw      0e317h  ; turn on byte mode
vpend   label   word
_DATA   ends
;
; Macro to output a word value to a port.
;
OUT_WORD        macro
if WORD_OUTS_OK
        out     dx,ax
else
        out     dx,al
        inc     dx
        xchg    ah,al
        out     dx,al
        dec     dx
        xchg    ah,al
endif
        endm
;
_TEXT   segment byte public 'CODE'
        assume  cs:_TEXT, ds:_DATA
;
; Sets up 360x480 256-color mode.
```

2

```
; (Courtesy of John Bridges.)
;
; Call as: void Set360By480Mode()
;
; Returns: nothing
;
        public _Set360x480Mode
_Set360x480Mode proc    near
        push    si              ;preserve C register vars
        push    di
        mov     ax,12h          ; start with mode 12h
        int     10h             ; let the bios clear the video memory

        mov     ax,13h          ; start with standard mode 13h
        int     10h             ; let the bios set the mode

        mov     dx,3c4h         ; alter sequencer registers
        mov     ax,0604h        ; disable chain 4
        out     dx,ax

        mov     ax,0100h        ; synchronous reset
        out     dx,ax           ; asserted
        mov     dx,3c2h         ; misc output
        mov     al,0e7h         ; use 28 mHz dot clock
        out     dx,al           ; select it
        mov     dx,3c4h         ; sequencer again
        mov     ax,0300h        ; restart sequencer
        out     dx,ax           ; running again

        mov     dx,3d4h         ; alter crtc registers

        mov     al,11h          ; cr11
        out     dx,al           ; current value
        inc     dx              ; point to data
        in      al,dx           ; get cr11 value
        and     al,7fh          ; remove cr0 -> cr7
        out     dx,al           ;    write protect
        dec     dx              ; point to index
        cld
        mov     si,offset vptbl
        mov     cx,((offset vpend)-(offset vptbl)) shr 1
@b:     lodsw
        out     dx,ax
        loop    @b
        pop     di          ;restore C register vars
        pop     si
        ret
_Set360x480Mode endp
;
; Draws a pixel in the specified color at the specified
; location in 360x480 256-color mode.
;
; Call as: void Draw360x480Dot(int X, int Y, int Color)
;
; Returns: nothing
;
DParms  struc
        dw      ?       ;pushed BP
        dw      ?       ;return address
DrawX   dw      ?       ;X coordinate at which to draw
DrawY   dw      ?       ;Y coordinate at which to draw
Color   dw      ?       ;color in which to draw (in the
                        ; range 0-255; upper byte ignored)
DParms  ends
;
        public _Draw360x480Dot
_Draw360x480Dot proc    near
        push    bp      ;preserve caller's BP
        mov     bp,sp   ;point to stack frame
        push    si      ;preserve C register vars
        push    di
        mov     ax,VGA_SEGMENT
        mov     es,ax   ;point to display memory
        mov     ax,SCREEN_WIDTH/4
```

Code continues . . .

PRODUCED 2003 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

S O F T W A R E S E C U R I T Y

WHETHER REPORT.

Whether you're a software developer writing new applications for the IBM or Mac, or a PC user securing proprietary data files, software and data protection has never had a brighter silver lining. For a number of very good reasons.

Beginning with the 'whether-expert' Rainbow Technologies. And ending with its Software Sentinel family of hardware keys. Starring five models that fit virtually any software program or data file you need to protect.

There's the best-selling SentinelPro for the IBM PC/XT/AT, PS/2 and compatibles, and even the Atari ST. Known worldwide for its virtually unbreakable security. And its ASIC technology. And its invisible operation. A close relation, the Sentinel-C stands at-the-ready for custom configurations and multiple software packages.

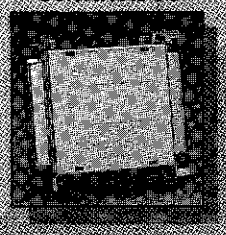
In the Apple market, security-minded Mac software developers turn to Eve. For completely transparent operation and world-class security of the protected software. Just by plugging Eve into the Mac ADB connector.

PC users wanting a low cost, user-friendly solution to the problem of securing sensitive data can call on the DataSentry. Using a proprietary Rainbow algorithm or DES, the DataSentry encrypts data files on individual PCs, protects modem transmissions and secures data on local area networks.

Rainbow's latest protection strategy is the SentinelShell – that lets users place a 'shell' around existing, off-the-shelf programs. Because access can be limited to those issued a key, libraries, universities and corporations can very simply guard their software investments.

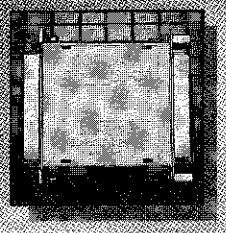
Whatever your whether, Rainbow Technologies has the software and data protection products that make the difference. For more information, call 714-261-0228 in the U.S., or contact Rainbow Technologies Ltd. in the United Kingdom for the distributor nearest you. Whethercasters are standing by.

SentinelPro™



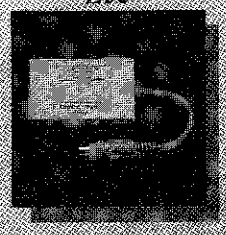
• *Runs under DOS, OS/2 and Xenix* • *Algorithm technique (Never a fixed response)* • *External parallel port installation* • *Minimal implementation effort* • *Higher level language interfaces included* • *100 times faster than fixed response devices (1 mS)* • *ASIC design for reliability*

Sentinel-C™



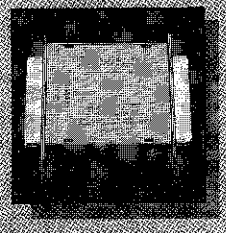
• *Protects multiple packages with one device* • *128 bytes of non-volatile memory pre-programmed before shipment of the software* • *Rainbow supplies a unique adapter for programming the unit* • *Higher level language interfaces included* • *Runs under DOS, OS/2 and Xenix* • *External parallel port installation*

Eve™



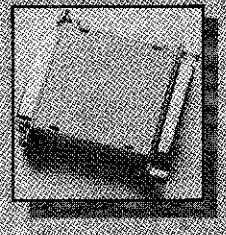
• *For the Macintosh SE and II* • *Complies with Apple Desktop Bus Interface requirements* • *Rainbow-assigned developer passwords to prevent tampering by other developers or sophisticated 'hackers'* • *7 locks per key, usable individually or in combination, on one or up to seven applications*

DataSentry™



• *Completely user-installable* • *Pocket-sized external device* • *Menu-driven, user-friendly interface* • *Single- or multi-user security system* • *Audit trail, log-on identifiers and automatic encryption/decryption of entire directories* • *Secures data transmitted by modems* • *Prevents recovery of data by utility programs*

SentinelShell™



• *Runs under DOS on IBM PCs and compatibles* • *Protects without requiring access to the source code* • *Completely transparent to the end user* • *User friendly software* • *Pocket-size key attaches quickly to any standard PC parallel port* • *ASIC design for reliability*

RAINBOW TECHNOLOGIES

18011-A Mitchell South, Irvine, CA 92714 • (714) 261-0228 • TELEX: 386078 • FAX: (714) 261-0260
Rainbow Technologies, Ltd., Shirley Lodge, 470 London Rd., Slough, Berkshire, SL3 8QY, U.K., Tel: 0753-41512, Fax: 0753-43610

© 1989 Rainbow Technologies. All product names are trademarks of their respective manufacturers.

Number **222** on the Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Abrash code, continued

3

```
                        ;there are 4 pixels at each address, so
                        ; each 360-pixel row is 90 bytes wide
                        ; in each plane
        mul     [bp+DrawY] ;point to start of desired row
        mov     di,[bp+DrawX] ;get the X coordinate
        shr     di,1    ;there are 4 pixels at each address
        shr     di,1    ; so divide the X coordinate by 4
        add     di,ax   ;point to the pixel's address
        mov     cl,byte ptr [bp+DrawX] ;get the X coordinate again
        and     cl,3    ;get the plane # of the pixel
        mov     ah,1
        shl     ah,cl   ;set the bit corresponding to the plane
                        ; the pixel is in
        mov     al,MAP_MASK
        mov     dx,SC_INDEX
        OUT_WORD        ;set to write to the proper plane for
                        ; the pixel
        mov     al,byte ptr [bp+Color]  ;get the color
        stosb           ;draw the pixel
        pop     di      ;restore C register vars
        pop     si
        pop     bp      ;restore caller's BP
        ret
_Draw360x480Dot endp
;
; Reads the color of the pixel at the specified
; location in 360x480 256-color mode.
;
; Call as: int Read360x480Dot(int X, int Y)
;
; Returns: pixel color
;
RParms  struc
        dw      ?       ;pushed BP
        dw      ?       ;return address
ReadX   dw      ?       ;X coordinate from which to read
ReadY   dw      ?       ;Y coordinate from which to read
RParms  ends
;
        public _Read360x480Dot
_Read360x480Dot proc    near
        push    bp      ;preserve caller's BP
        mov     bp,sp   ;point to stack frame
        push    si      ;preserve C register vars
        push    di
        mov     ax,VGA_SEGMENT
        mov     es,ax   ;point to display memory
        mov     ax,SCREEN_WIDTH/4
                        ;there are 4 pixels at each address, so
                        ; each 360-pixel row is 90 bytes wide
                        ; in each plane
        mul     [bp+DrawY] ;point to start of desired row
        mov     si,[bp+DrawX] ;get the X coordinate
        shr     si,1    ;there are 4 pixels at each address
        shr     si,1    ; so divide the X coordinate by 4
        add     si,ax   ;point to the pixel's address
        mov     ah,byte ptr [bp+DrawX]
                        ;get the X coordinate again
        and     ah,3    ;get the plane # of the pixel
        mov     al,READ_MAP
        mov     dx,GC_INDEX
        OUT_WORD        ;set to read from the proper plane for
                        ; the pixel
        lods    byte ptr es:[si] ;read the pixel
        sub     ah,ah   ;make the return value a word for C
        pop     di      ;restore C register vars
        pop     si
        pop     bp      ;restore caller's BP
        ret
_Read360x480Dot endp
_TEXT   ends
        end
```

4

LISTING 2

```
/*
 * Sample program to illustrate VGA line drawing in 360x480
 * 256-color mode.
 *
 * Compiled with Turbo C 2.0.
 *
 * Must be linked with Listing 1 with a command line like:
 *
 *    tcc l2 l1.asm
 *
 * By Michael Abrash.  6/2/89.
 */
#include <dos.h>                /* contains geninterrupt */

#define TEXT_MODE       0x03
#define BIOS_VIDEO_INT  0x10
#define X_MAX           360     /* working screen width */
#define Y_MAX           480     /* working screen height */

extern void Draw360x480Dot();
extern void Set360x480Mode();

/*
 * Draws a line in octant 0 or 3 ( |DeltaX| >= DeltaY ).
 * |DeltaX|+1 points are drawn.
 */
void Octant0(X0, Y0, DeltaX, DeltaY, XDirection, Color)
unsigned int X0, Y0;            /* coordinates of start of the line */
unsigned int DeltaX, DeltaY;    /* length of the line */
int XDirection;                 /* 1 if line is drawn left to right,
                                   -1 if drawn right to left */
int Color;                      /* color in which to draw line */
{
   int DeltaYx2;
   int DeltaYx2MinusDeltaXx2;
   int ErrorTerm;

   /* Set up initial error term and values used inside drawing loop */
   DeltaYx2 = DeltaY * 2;
   DeltaYx2MinusDeltaXx2 = DeltaYx2 - (int) ( DeltaX * 2 );
   ErrorTerm = DeltaYx2 - (int) DeltaX;

   /* Draw the line */
   Draw360x480Dot(X0, Y0, Color);   /* draw the first pixel */
   while ( DeltaX-- ) {
        /* See if it's time to advance the Y coordinate */
        if ( ErrorTerm >= 0 ) {
           /* Advance the Y coordinate & adjust the error term
              back down */
           Y0++;
           ErrorTerm += DeltaYx2MinusDeltaXx2;
        } else {
           /* Add to the error term */
           ErrorTerm += DeltaYx2;
        }
        X0 += XDirection;           /* advance the X coordinate */
        Draw360x480Dot(X0, Y0, Color);    /* draw a pixel */
   }
}

/*
 * Draws a line in octant 1 or 2 ( |DeltaX| < DeltaY ).
 * |DeltaY|+1 points are drawn.
 */
void Octant1(X0, Y0, DeltaX, DeltaY, XDirection, Color)
unsigned int X0, Y0;            /* coordinates of start of the line */
unsigned int DeltaX, DeltaY;    /* length of the line */
int XDirection;                 /* 1 if line is drawn left to right,
                                   -1 if drawn right to left */
int Color;                      /* color in which to draw line */
{
   int DeltaXx2;
```

Code continues . . .

PRODUCED 2003 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

New!

TurboGeometry-Plus

A Library of 2D and 3D Geometric Routines

Over 300 2D & 3D geometric routines. Includes all the routines in TurboGeometry Library plus • Surfacing • Solids • Additional Hide Line Algorithms • Spherical Geometry • 3D Clipping • and more. Easy to use manual. 30 day guarantee. $199.95 or $299.95 with source. Foreign Orders $225.00 or $325.00 with source. Shipping & Handling included.

TurboGeometry Library 3.0

NEW VERSION, 3.0 Over 200 routines covering • Intersections • Volumes • Areas • Transforms • Perspectives • Decomposition • Tangents • Clipping • and more. Easy to use manual, 30 day guarantee. $149.95 or $199.95 with source. Foreign Orders $175.00 or $225.00 with source. Shipping and Handling included.

Both products available in Turbo Pascal, Turbo C, Microsoft C or Turbo Pascal Macintosh. Need MS/PC DOS 2.0+, IBM PC or Compatible, Macintosh. Visa/MC, PO or Chk on a USA Bank.

Disk Software, Inc.

2116 E. Arapaho Rd.
Suite 487
Richardson, TX 75081
USA

214-423-7288
214-423-4465 FAX
800-635-7760

Number **229** on the Reader Service Card

THE ULTIMATE PROTECTION

StopCopy Plus and StopView:
Quite Simply, the Best Ways to Protect Your Valuable Software Investment.

StopCopy Plus:
Sophisticated Copy Protection that defeats all hardware/software copiers, is loaded with developer options, and remains completely transparent to the honest end user.

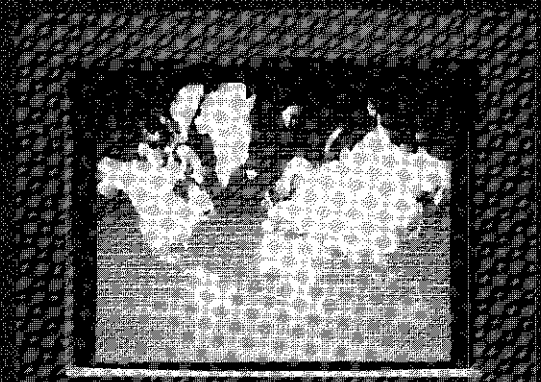
StopView:
Advanced copy deterrent that stops hackers, allows tracing of your software, and provides many options without copy protection.

Free demonstration disk available!
Call Today!

BBI Computer Systems
14105 Heritage Lane
Silver Spring, MD 20906
Phone: (301) 871-1094 Fax: (301) 460-7545

Number **303** on the Reader Service Card

CUT PROGRAMMING TIME. WITH GRAPHIC RESULTS.



Get to the market quicker with graphics that don't compromise performance. GSS® Graphics Development Toolkits for DOS and OS/2 give you access to more than 100 high-level graphics functions. And your GDT-based applications will support over 300 graphics devices.

The GDT is a time-proven solution for programs written in C, FORTRAN, Pascal, BASIC Compiler or Macro Assembler. It has been optimized through eight years of refinement and application to produce high-quality, high-performance graphics.

YOU'LL BE IN GOOD COMPANY.

IBM has licensed the GDT for its three PC operating systems: DOS, OS/2 and AIX. SPC's Harvard Graphics and Ashton-Tate's Draw APPLAUSE use GDT technology. So do hundreds of other PC packages for science, business and engineering.

IN A HURRY?

CALL (503) 641-2455.

ASK FOR DEPARTMENT PJ

Order your GDT, or ask for a free copy of our Devices Supported booklet. The GDT is the most direct route to completing your applications—with a graphic difference.

The GSS logo and GSS are registered trademarks of Graphic Software Systems, Inc.

Graphic Software Systems

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Abrash code, continued

5

```
   int DeltaXx2MinusDeltaYx2;
   int ErrorTerm;

   /* Set up initial error term and values used inside drawing loop */
   DeltaXx2 = DeltaX * 2;
   DeltaXx2MinusDeltaYx2 = DeltaXx2 - (int) ( DeltaY * 2 );
   ErrorTerm = DeltaXx2 - (int) DeltaY;

   Draw360x480Dot(X0, Y0, Color);     /* draw the first pixel */
   while ( DeltaY-- ) {
      /* See if it's time to advance the X coordinate */
      if ( ErrorTerm >= 0 ) {
         /* Advance the X coordinate & adjust the error term
            back down */
         X0 += XDirection;
         ErrorTerm += DeltaXx2MinusDeltaYx2;
      } else {
         /* Add to the error term */
         ErrorTerm += DeltaXx2;
      }
      Y0++;                  /* advance the Y coordinate */
      Draw360x480Dot(X0, Y0,Color);  /* draw a pixel */
   }
}

/*
 * Draws a line on the EGA or VGA.
 */
void EVGALine(X0, Y0, X1, Y1, Color)
int X0, Y0;     /* coordinates of one end of the line */
int X1, Y1;     /* coordinates of the other end of the line */
unsigned char Color;     /* color in which to draw line */
{
   int DeltaX, DeltaY;
   int Temp;

   /* Save half the line-drawing cases by swapping Y0 with Y1
      and X0 with X1 if Y0 is greater than Y1. As a result, DeltaY
      is always > 0, and only the octant 0-3 cases need to be
      handled. */
   if ( Y0 > Y1 ) {
      Temp = Y0;
      Y0 = Y1;
      Y1 = Temp;
      Temp = X0;
      X0 = X1;
      X1 = Temp;
   }

   /* Handle as four separate cases, for the four octants in which
      Y1 is greater than Y0 */
   DeltaX = X1 - X0;     /* calculate the length of the line
                            in each coordinate */
   DeltaY = Y1 - Y0;
   if ( DeltaX > 0 ) {
      if ( DeltaX > DeltaY ) {
         Octant0(X0, Y0, DeltaX, DeltaY, 1, Color);
      } else {
         Octant1(X0, Y0, DeltaX, DeltaY, 1, Color);
      }
   } else {
      DeltaX = -DeltaX;              /* absolute value of DeltaX */
      if ( DeltaX > DeltaY ) {
         Octant0(X0, Y0, DeltaX, DeltaY, -1, Color);
      } else {
         Octant1(X0, Y0, DeltaX, DeltaY, -1, Color);
      }
   }
}

/*
 * Subroutine to draw a rectangle full of vectors, of the
 * specified length and in varying colors, around the
 * specified rectangle center.
```

6

```
 */
void VectorsUp(XCenter, YCenter, XLength, YLength)
int XCenter, YCenter;   /* center of rectangle to fill */
int XLength, YLength;   /* distance from center to edge
                           of rectangle */
{
   int WorkingX, WorkingY, Color = 1;

   /* Lines from center to top of rectangle */
   WorkingX = XCenter - XLength;
   WorkingY = YCenter - YLength;
   for ( ; WorkingX < ( XCenter + XLength ); WorkingX++ )
      EVGALine(XCenter, YCenter, WorkingX, WorkingY, Color++);

   /* Lines from center to right of rectangle */
   WorkingX = XCenter + XLength - 1;
   WorkingY = YCenter - YLength;
   for ( ; WorkingY < ( YCenter + YLength ); WorkingY++ )
      EVGALine(XCenter, YCenter, WorkingX, WorkingY, Color++);

   /* Lines from center to bottom of rectangle */
   WorkingX = XCenter + XLength - 1;
   WorkingY = YCenter + YLength - 1;
   for ( ; WorkingX >= ( XCenter - XLength ); WorkingX-- )
      EVGALine(XCenter, YCenter, WorkingX, WorkingY, Color++);

   /* Lines from center to left of rectangle */
   WorkingX = XCenter - XLength;
   WorkingY = YCenter + YLength - 1;
   for ( ; WorkingY >= ( YCenter - YLength ); WorkingY-- )
      EVGALine(XCenter, YCenter, WorkingX, WorkingY, Color++);
}

/*
 * Sample program to draw four rectangles full of lines.
 */
int main()
{
   char temp;

   Set360x480Mode();

   /* Draw each of four rectangles full of vectors */
   VectorsUp(X_MAX / 4, Y_MAX / 4, X_MAX / 4, Y_MAX / 4, 1);
   VectorsUp(X_MAX * 3 / 4, Y_MAX / 4, X_MAX / 4, Y_MAX / 4, 2);
   VectorsUp(X_MAX / 4, Y_MAX * 3 / 4, X_MAX / 4, Y_MAX / 4, 3);
   VectorsUp(X_MAX * 3 / 4, Y_MAX * 3 / 4, X_MAX / 4, Y_MAX / 4, 4);

   /* Wait for the enter key to be pressed */
   scanf("%c", &temp);

   /* Back to text mode */
   _AX = TEXT_MODE;
   geninterrupt(BIOS_VIDEO_INT);
}
```

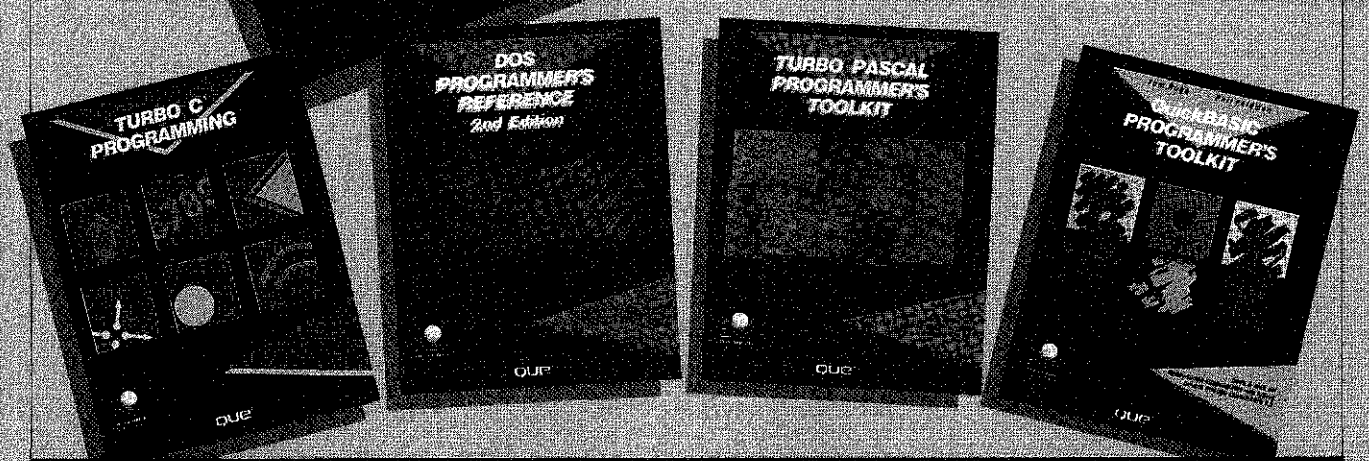

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Graphic Results!

Only $24.95

Maximize your EGA/VGA programming graphics with Que's *Power Graphics Programming!* Que has teamed up with *Programmer's Journal* to bring you this nuts-and-bolts graphics programming guide. Now, for the first time, you can have the expertise of Michael Abrash consolidated in a single volume. His insights and approach to efficient, effective programming have earned him high acclaim for his "On Graphics" series. If you are seeking that elusive graphic edge, you can't be without *Power Graphics Programming!*

Available in better bookstores and computer stores everywhere. Or call 800-428-5331, ext. PJ01.



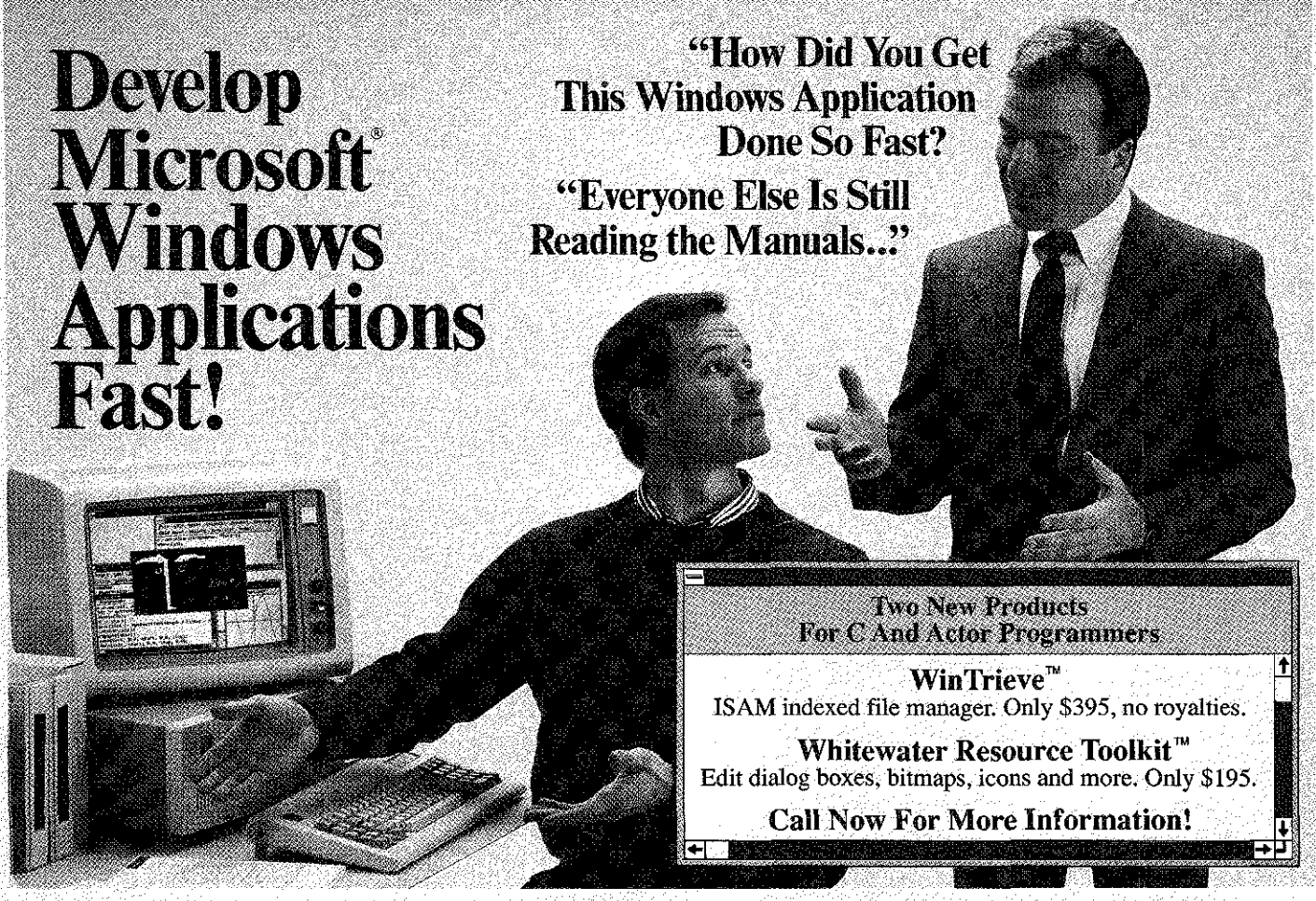
Look for these other practical Programming books from Que

que®

Number 295 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Develop Microsoft® Windows Applications Fast!


"How Did You Get This Windows Application Done So Fast?
"Everyone Else Is Still Reading the Manuals..."
Two New Products For C And Actor Programmers
WinTrieve™
ISAM indexed file manager. Only $395, no royalties.
Whitewater Resource Toolkit™
Edit dialog boxes, bitmaps, icons and more. Only $195.
Call Now For More Information!

The Windows Advantage

You've selected Microsoft Windows as your graphical user interface. Windows applications are easier for end-users to learn and use. It is **the** standard for PCs.

Now you want Windows applications completed as fast as possible. But it takes time for your programmers to get up to speed on the intricacies of Windows development.

Why The Delay For Windows Applications?

Programming for the Windows environment is very different from character-based programming. Just learning the 450 Windows function calls can take months. Getting used to its event-driven system and memory management takes additional time.

Programmers who are new to Windows development are looking for support to help them produce high-quality Windows programs quickly and efficiently.

3 Ways To Speed Up Windows Application Development

Now there are specific ways to speed up applications development under Windows.

- **Code reduction.** There is a fast, effective way to develop Windows applications. A way that cuts the amount of code needed. A way that by-passes much of the complexity of Windows programming. A way that doesn't compromise performance.
- **Short learning curve.** There is an easy, convenient way to get your entire programming staff up to speed with Windows within weeks.
- **From prototype to complete application.** There is a way to quickly develop prototypes of Windows applications in days, not months. Then, turn those into completed applications quickly.

1 Product, 3 Solutions

There is **one** product which brings **all three** solutions to your Windows programmers. It's called **Actor.®**

Actor is a proven product. It's now being used in thousands of software development groups world-wide.

Actor Does Windows

Actor gives your programmers a fast, effective way to develop Windows applications. **Actor's** complete development environment allows you to produce everything from prototypes through stand-alone applications. Programs developed in **Actor** are fast and support all Windows features.

Actor offers interactive source code debugging and testing—greatly reducing development time. Your programmers benefit from high code reusability because **Actor** is an object-oriented programming language. This allows them to build on existing code, eliminating duplication of effort. They can even dynamically link to existing C code.

Actor also provides one more essential: fast, responsive support from The Whitewater Group,® recognized worldwide as leaders in Windows programming.

If you are programming applications for Windows, you should be using **Actor.** Users report that **Actor** cuts Windows programming time by at least 50%.

Act Now!

See for yourself how **Actor** can help your Windows programmers! The first step is to order our free booklet, **For Faster, Better, Smaller, Smarter Windows Programming.** Examine detailed information about using **Actor,** complete with actual source code examples.

Call us at 312/328-3800. We'll send you the booklet and complete details about **Actor.**

Take 60 seconds now to save months of development time in the future.

The Whitewater Group®
600 Davis Street
Evanston, IL 60201
(312) 328-3800

Microsoft is a registered trademark of Microsoft Corporation. Actor is a registered trademark, The Whitewater Group is a registered servicemark, and WinTrieve and Whitewater Resource Toolkit are trademarks of The Whitewater Group, Inc.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Forest for the Trees

by

Vaughn Vernon

While I have concentrated on writing code in the past, this article will focus more on an issue that will affect your C++ development: the style of class libraries you purchase and develop. Although you may not realize it, C++ allows you to use and develop software in several different styles. In fact, you can even use C++ as a non-object-oriented language. Nevertheless, I will continue to discuss how C++ fits into object-oriented programming and also describe two different approaches to class libraries.

The Issues

A very big issue in object-oriented programming is cooperation—how does one object live with and relate to another to accomplish the ultimate goal? Can objects of various classes coexist and share resources without problems?

Exception handling is another extremely important issue. In C++, exceptions can occur with all the dynamic "new" memory being allocated and freed on the fly. If you came into a function expecting to declare and use a particular object, but no memory existed to allocate the object, a program without exception handling would continue to ramble on until core dumped or the machine locked up. However, with a robust exception-handling mechanism, you would be more likely to recover from the error and exit elegantly.

The last, but definitely not least, issue is ease of use. Whether or not a class library is easy to use will determine its ultimate success or failure. Ease of use is quite possibly more important than the other two issues.

Existing Work

If you have used C++ at all, you know that C++ class libraries are not widely available to mere mortals. I have talked to several folks from AT&T Bell Labs who relate the joy of using this-class or that-class library, but I have yet to see one packaged with C++. There are some public domain offerings, but, as you well know with public domain, "what you seize is what you gets()!" So, in order to discuss the theories behind the development and use of class libraries, I may at times have to refer to offerings from other OO languages.

One of the public domain offerings is the OOPS library from Keith Gorlen of the National Institutes of Health. This set of C++ classes offers a Smalltalk-like library. Smalltalk provides a class-rich programming environment that includes facilities for managing anything from bit vectors to windows. But for our discussion, I want to focus on *how* Smalltalk and, hence, the OOPS library implement its classes.

The Smalltalk approach to classes is via a class-hierarchy tree. (Many other popular OO systems, such as Eiffel, C_Talk, and Objective-C, use the Smalltalk approach.) At the class-tree root is the class Object. The tree grows from the root as new classes inherit from Object. As a result, all new classes inherit from Object no matter how far from the root they live. A partial class tree is shown in Figure 1.

In Smalltalk, when a new class is needed, and that class is truly unique and does not need to inherit from higher-level classes, it must at least inherit from Object. So, Blip would be derived from Object in the OOPS library:

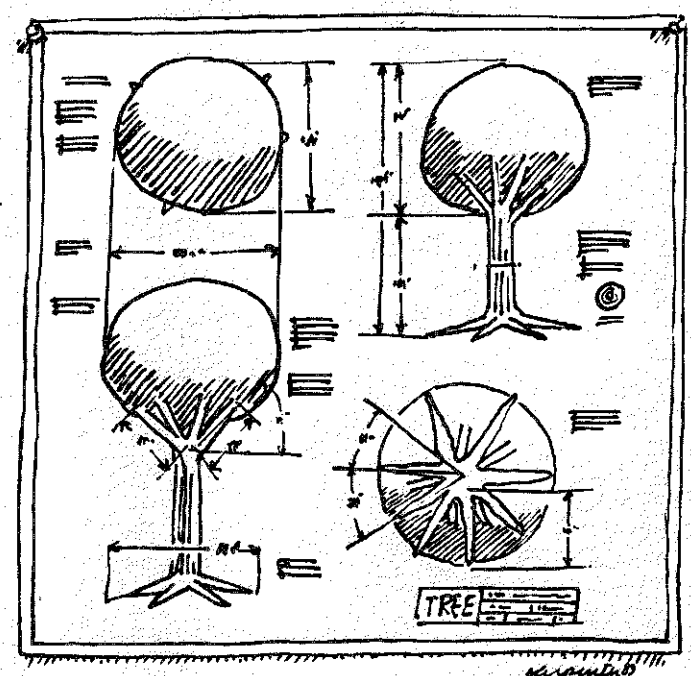
```
class Blip : public Object { // ... };
```

By inheriting from Object, the class is given features that are necessary for life in the Smalltalk (and, hence, the OOPS library) environment. You will see why a little later.

On the other hand, most class libraries that I have seen or read about for the PC (and other) implementations of C++ have taken on a form different from the Smalltalk approach. These class libraries do not promote a tree structure of classes that grow from a root class. When a new class is needed that is unique and does not *require* features from an existing class, the class is created as a new root of its own:

```
class Blip { // ... };
```

If another class needs the features of the Blip class, it inherits from Blip, which continues the growth of the new tree. Since



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

so many of these trees are usually found in a class library, it is appropriate to refer to a "forest" of C++ classes (see Figure 2).

I cannot take credit for coining the forest description. I'm not sure who thought of it first, but I first heard the term at the 1988 Usenix C++ Conference in Denver, where it was used by Douglas Lea to describe the structure (or lack thereof) in the libg++ class library. (The libg++ project was sponsored by Project GNU.) In his paper, Mr. Lea described his forest approach as a "less ambitious-looking approach."

A Tree or a Forest; A ++ or -- ?

Is there a definite advantage to using a tree approach or a forest approach? The answer may be yes or no. It really depends on what you want to accomplish and what you expect out of your class library. Let's consider some advantages and disadvantages of both approaches.

First off, if you examine the very nature of the tree approach, you get an idea of the advantages available. Since all classes have a common ancestor, Object, all classes *are* objects. That means that all objects of any class can be created, collected, compared and generally manipulated as objects, and each object knows its real class. You automatically inherit class methods that operate on generic objects. Also, new class methods can be written in a generic way by taking arguments of type Object, Object *, or Object &, since *most* arguments are Objects. All that means code reuse.

For instance, say you wish to create a Collection class. A Collection is basically an array of arbitrary objects. However, since all elements of the Collection are ultimately Objects, the Collection class requires no special-case code to deal with special-case classes. Additionally, if you added the class method, you could ask a generic object to print its class tree, and it could show you the path all the way back to Object!

Similarly, since all classes inherit from Object, implementing an exception-handler mechanism for all classes is easy. You would place the exception-handler protocol in the Object class, and all new classes would inherit the feature. William Miller of Software Development Technologies, Inc., contributed a nice solution.

Now for the bad news: Unfortunately, not everything in C++ (as of v1.2) is an Object. Take the standard C data types—**int**, **char**, **float**, **long**, and so forth. If you want a Collection of **int** variables, you're up a creek. And if you plan on supporting your own Integer, Float, or Character classes, have fun—I'm sure that such an effort would lead to performance pains and other eventual headaches. This is a problem that should be solved by the proposed C++ v2.0 parameterized types.

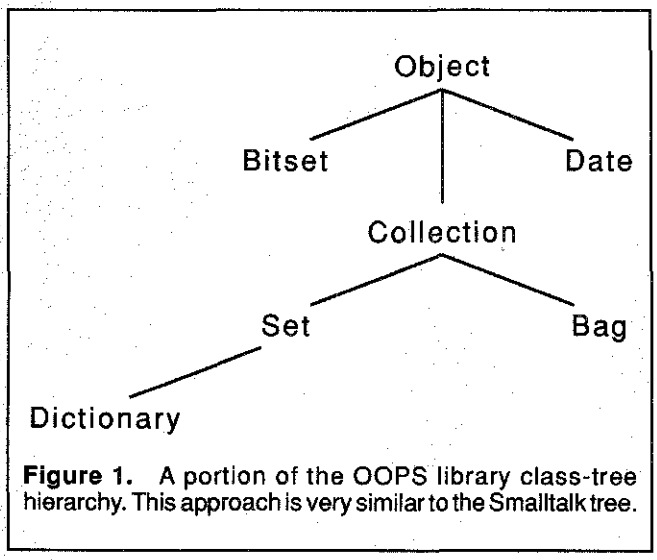


Figure 1. A portion of the OOPS library class-tree hierarchy. This approach is very similar to the Smalltalk tree.

Additionally, those whom I have spoken to about the OOPS library have found it hard to use. They claim that when you want to derive a new class, there are "so many virtual methods" that reimplementing them for each new class is a job in itself. It takes some time before you start solving the problem you set out to solve. This is not necessarily a reflection on Gorlen's work but, rather, a reflection on what he had to do to tip-toe (or possibly stomp) around things that the current version of C++ was never designed to handle. However, I imagine that the problem could be solved with a few compromises yielding a lot more simplicity and, therefore, ease of use.

One parting thought on the OOPS library: It was developed on UNIX with obvious lack of regard for DOS limitations; it was never meant to be portable. I have a copy of it in DOS format, but things like long filenames and the overall size and complexity of the library make it difficult to impossible to use under DOS. I'm sure no translator could survive its demands for memory, and the Zortech compiler would not handle its syntax. This really has nothing to do with a class-tree problem, but it may save you from rushing out to get a copy of it.

As you will see, the forest approach has its own advantages and disadvantages. Oddly enough, the advantages of one approach seem to highlight the disadvantages of the other, and vice versa. Since a forest library inherits only when necessary, you don't end up reimplementing virtual methods defined in class Object for each and every class. This tends to make problem solving straightforward and as fast as possible.

However, the Collection class discussed earlier cannot be written as elegantly using the forest method. You must use an alternative, kludged approach, since no one object can be guaranteed to be related to any other. I do not have the space for details here, but you can find a discussion of it in the Stroustrup book (see references) on page 209, under the heading "Generic Classes." This method also lacks a high degree of code reuse.

The forest approach is also lacking in its ability to couple well with the exception-handling mechanism mentioned earlier. That is because you are not forced to inherit from a single common class, which would contain the exception-handling protocol.

All in all, the forest approach seems to be the easiest to use, at least for beginners. It has the approval of many in the C++ community. However, you sure pass up a lot by overlooking the tree approach. To me, at least, this would be the ultimate approach if some needed features where added to C++.

Which Approach Is Better for You?

It depends on what you want to accomplish. If exception handling is a must, then the tree approach fits your needs better, at least at this stage of C++ infancy. However, the forest approach is good for adding a veneer over an existing C library. For instance, if you were to create a C++ class interface to an existing tool such as a database library, wrapping the func-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

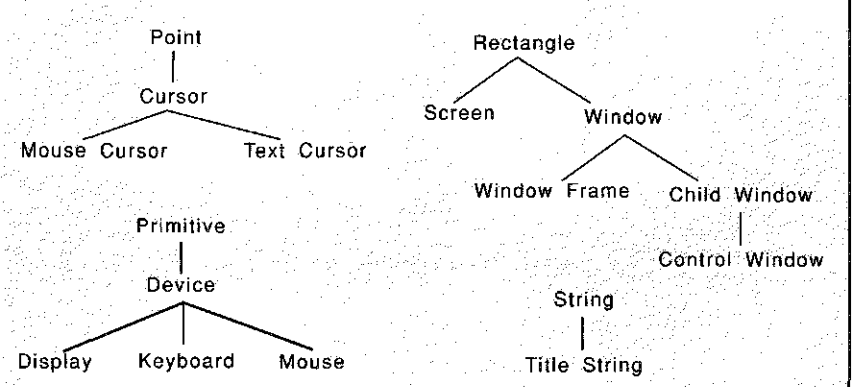


Figure 2. A possible windowing library using the "forest" approach. Classes are created as needed, without regard for structure.

It is obvious to me that C++ needs a standard class library, just as C needed a standard C library, and it should be defined by the language implementors (let's not give it to ANSI just yet). It does not need to be very elaborate—just the basics: Collection, Stack, Date, Time, InputFile, OutputFile, and the like, probably the same basic classes the OOPS library contains, but easier to use, small, and portable. In addition, the exception-handling protocol would be part of the Object class. This would at least give those who are developing specialty class libraries a basis with which to start, and it wouldn't limit you from adding multiple inheritance to the branches of the tree.

tion interface with a set of classes and in-line methods would be fast and easy using the forest approach.

If you attempted to use a tree and a forest library at the same time, they would not cooperate well with one another. Clearly, if you commit to either of these approaches, it could lock you out from using future tools. For instance, what if AT&T chooses to bundle a "Standard C++ Class Library" with the translator starting in version 2.0? If their library were a tree type, and you committed to a forest approach now, using a standard library later could be tough—a problem that probably prevents most companies from switching from C to C++ right now.

Cfront: Out of Memory

In my last article (*PJ* 7.3), I talked about the problems most DOS-based C++ translators have with memory. I would like to get something straight here: When I criticize a memory problem, I am not criticizing the people who ported a program to DOS. Although some want to argue over details, others are willing to listen to solutions. Nevertheless, it is a problem. To deny it would be an injustice to my readers. Cfront, the C++ translator, is a huge program, and besides, it does a lot of memory allocation on top of that! To translate even the seemingly simplest source can be a nightmare. And running MAKE to build your program is out of the question.

HARVARD SOFTWORKS

NUMBER ONE IN FORTH INNOVATION

(513) 748-0390 P.O. Box 69, Springboro, OH 45066

Have you ever wanted a language that:

- Combined the raw power of extensible languages with the convenience of carefully implemented functions?
- Executed faster than optimized C?
- Compiled 20,000 lines per minute?
- Was totally interactive, even while compiling?
- Helped you program at any level of of abstraction from machine code thru application specific language with equal ease and efficiency?
- Allowed easy interactive modification of any routine at any time?
- Gave source code for 2000 functions?
- Could borrow any data structure, any control structure, any interface scheme from any other language?
- Could implement the borrowed feature just as efficiently, and often more efficiently, than its source?
- Had an architecture that could support very small programs and full megabyte ones with a single version?
- Lacked mystical syntax requirements?
- Could use subroutine libraries written for other languages? More efficiently?
- Enabled novice programmers to run rings around Computer Science graduates?

Forth has to often been the language that tempted programmers with "great expectations", then frustrated them beyond all endurance with the need to reinvent the simplest tools expected in any commercial language.

HS/FORTH is different!

We won't short change you with "great possibilities". We will provide the implemented functionality to let you complete your application quickly.

You can't judge the power of Forth by public domain products or ones from vendors more interested in consulting. The latter earn more by not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful in learning the basic nature of Forth, they fail at showing its true potential. Not to mention being s-l-o-w.

You can't add extensibility to closed compilers. You can add features from any other language to **HS/FORTH**. We even added full **OOPs** capability. **HS/FORTH** has more features and libraries than we can possibly describe here. Please ask for our brochure.

HS/FORTH for MSDOS or PCDOS. Each higher level includes all features of the lower ones. Level upgrades: $25. plus price difference. Source code in normal ASCII text files.

STUDENT LEVEL	**$145.**
text & scaled/clipped graphic windows; powerful parsing, formatting, file and device I/O; shells; interrupt handlers	
PERSONAL LEVEL	**$245.**
software floating point, quad & scaled integer, automatic recursive optimizer	
PROFESSIONAL LEVEL	**$395.**
hardware floating point, turnkey, seal, Rosetta Stone Dynamic Linker, round robin & time slice multitaskers, dynamic string manager, file blocks, sector mapped blocks, assemblers	
PRODUCTION LEVEL	**$495.**
metacompilation for DOS or ROM, full system ROMability, direct compilation of C data structures, MetaWindows graphics library	

Options:

FOOPS object oriented Forth	**$ 45.**
GigaForth (80286 native mode)	**$295.**
TurboWindow-C (Metagraphics)	**$ 79.**
enhanced interface	**$ 29.**
BTRIEVE (Novell)	**$175.**
enhanced interface	**$ 29.**

Number **42** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

If you have a shortage of memory, I have two solutions for you:

- Buy an 80386 AT-compatible and put a least 2 megabytes of memory in it. Then do one of the following: Buy the C++ translator or add-on translator with a 386 DOS extender to take advantage of extended memory, or buy 386 UNIX and VP/IX or a UNIX cross-translator that creates DOS binaries. This will cost you about $4,000—maybe less if you shop around.
- Try out the MAKE program that I've included on this issue's code disk.

Although I just said you couldn't use MAKE to build your .EXE, I also forgot to tell you about my simple MAKE program. About five years ago, when I first began to fiddle around with my CURSES Window library, all I had to develop on was a 8086 PC with 256K of memory. I wrote a MAKE program because no one else had one, and I couldn't stand manual labor. Then I noticed that some programs were too large for MAKE to spawn. So I added a cute little feature: a command line switch to cause MAKE to generate a DOS batch file (.BAT) rather than try to run the commands directly. So after typing MAKE (about two seconds later), I just typed **mk** to run the batch file.

Well, CURSES started to sell, and I bought myself an AT and put 3.2 megabytes of memory in it (for OS/2). I no longer had a 256K limit. But, soon enough, I discovered that I now had a 640K limit when using C++. Back to the drawing board! I recently modified my MAKE to support the various C++ translators I own. Although you may still find memory limited, at least you can use MAKE again.

If you are into trying a C++ for 386 DOS extended memory, this translator may be for you. A company called Intek (Integration Technologies, 1400 112th Street S.E., Suite 202, Bellevue, WA 98004, 206-455-9935) uses C++ to develop custom software. They recently made available (for $495.00) an extended memory version for 386 machines. A big difference between the Intek offering and several of the others is that Intek supports other compilers in addition to Microsoft C— Turbo C and WATCOM for MS-DOS applications and MetaWare High C for 386-specific applications. PJ

Vaughn Vernon is president of Aspen Scientific, producers of CURSES and FORMATION, user-interface cross-development tools for UNIX, MS-DOS, OS/2, and VMS. He can be reached at P.O. Box 72, Wheatridge, CO 80034.

References

Gorlen, Keith E., "An Object-Oriented Class Library for C++ Programs," USENIX Proceedings, C++ Workshop, 1987.

Lea, Douglas, "libg++, The GNU C++ Library," Usenix Proceedings, C++ Workshop, 1988.

Miller, William M., "Exception Handling Without Language Extensions," Usenix Proceedings, C++ Workshop, 1988.

Stroustrup, Bjorne, *The C++ Programming Language*, Addison-Wesley, 1986.

What Is C++?

with William M. Miller

Evaluating C++?

There's no quicker, easier way to get a comprehensive overview of the language than this 90-minute VHS videotape.

Learning C++?

The task will be much easier if you get a firm grasp of the concepts before you get lost in the details.

Our moneyback guarantee means there's *no risk!* And Mr. Miller is available for followup questions via BIX. Order your copy today - $19.95 + $4 S&H for a 1-week rental, $59.95 postpaid for purchase - with Visa, MasterCard, check, or money order.

Software Development Technologies, Inc.
Department 403
P.O. Box 366, Sudbury, MA 01776-0003
(508) 443-5779

Number 275 on the Reader Service Card

4c™ Hypertext Source Code Databank for Programmers

Explore source code! Navigate uncharted logic trails easily!

4 c includes:

★ A Source Code Analyzer:
The analyzer maps and cross-references functions and variables to their respective path and file names. The cross-reference table can be displayed alphabetically and by type of variable. It allows a programmer to edit functions directly without specifying a file name.

★ A Point-and-Shoot Zoom Key:
Which instantly locates any function, global variable, structure definition, #define statement, or macro, etc., in any file, directory, and drive, and displays it in a new edit window.

★ A Full Featured Built in Editor:
Featuring multiple files, multiple windows, cut-and-paste between files, search-and-replace, auto indent, tab support, and a familiar user interface with pull-down menus, or use your own editor!

"Given a procedure's identifier, an editor should be able to bring up the full procedure interface, including parameters, so that you don't have to remember how many and in what order. Wouldn't it be nice to hit a hot key on the name of a complicated data type and have its definition pop up in a window? It may seem like a real small matter, but how many of you still flip through printouts doing things the editor should be doing?"

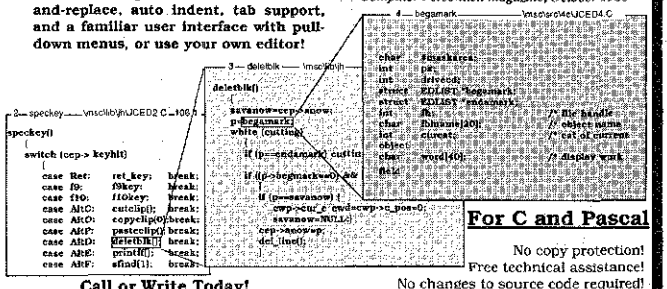
- Phillipe Kahn, "The Connection", Spring/Summer 1988

4 c does this and more

"4c's Magic is apparent..."
- PC Magazine, February 1989

"Daylight in the C Confusion"
- Toolbox magazine, December 1988

"...Destined to become the standard editor for C..."
- Computer Personlich magazine, October 1988



For C and Pascal

No copy protection!
Free technical assistance!
No changes to source code required!
Required IBM PC compatible with 350K RAM.
30 Day Money Back Guarantee!
Only $119.00 plus shipping
Visa, Mastercard, check, money order, or COD.

Call or Write Today!

Tri-Technology Systems Inc.
1225 S. Elgin, Forest Park, Il. 60130
Phone: 312-366-7595

Number 276 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MicroWay Means Numerics!

MicroWay is your best source for the software and hardware you need to get true 32 bit performance from your 386. These include 32-bit tools, such as NDP Fortran, C, and Pascal, and the 32-bit applications that were developed with them (see last paragraph). These products run in protected mode under Unix, Xenix, or Phar Lap extended MS-DOS.

Starting with release 1.4VM, NDP Fortran, C and Pascal not only access 4 gigabytes of memory, but run with Phar Lap's new VMM extension which provides 386 protected mode virtual memory. Now you can run a program with a 30 MB array on a 2 MB system simply by having 30 MB of space on your hard disk.

MicroWay also offers transputer based parallel processing boards and languages that run in an XT, AT, or 386. Each of the T800 RISC processors on these boards packs the power of a 20 MHz 386/1167. Our best selling board, the Quadputer2™, has four T800s and boasts 40 MIPS/6 megaflops of processor throughput.

MicroWay manufactures Weitek 1167 and 3167 coprocessor cards that run with the 80386. Both cards include an 80387 socket. The 1167 is 2 to 4 times faster than the 80387. The 3167 runs 30% faster than the 1167 in double precision. The key to achieving this speed increase is our NDP Fortran or C and the new 32-bit applications that offer Weitek support. Either processor provides a dramatic increase in throughput for graphics intensive applications. These include VersaCad and Hoops 3D graphics, ANVIL 5000 CAD/CAM, SRAC and Swanson Analysis finite element packages, Mathematica and a host of other packages that were recently ported to the 386 using our NDP Fortran and NDP C. **Please call (508) 746-7341 for more information.**

Dr. Robert Atwell, leading defense scientist, calculates that NDP Fortran-386 is saving him $12,000 per month in rentals of VAX hardware and software while doubling his productivity!

Fred Ziegler of AspenTech in Cambridge, Mass. reports, "I ported 900,000 lines of Fortran source in two weeks without a single problem!" AspenTech's Chemical Modeling System is in use on mainframes worldwide and is probably the largest application to ever run on an Intel processor.

Dr. Jerry Ginsberg of Georgia Tech reports, "My problems run a factor of six faster using NDP Fortran-386 on an mW1167 equipped 386/20 than they do on my MicroVAX II."

32-Bit Compilers and Tools

NDP Fortran-386™, **NDP C-386™**, and **NDP Pascal-386™** compilers generate globally optimized, mainframe quality code. Each runs in 386 protected mode under Unix, Xenix or Phar Lap extended MS-DOS. The memory model employed uses 2 segments, each of which can be up to 4 gigabytes. They generate code for the 80287, 80387, mW3167 or mW1167 and include high speed EGA graphics extensions written in C that perform BASIC-like screen operations. **NDP Fortran-386™** is a full implementation of FORTRAN-77 with Berkeley 4.2 and Fortran-66 extensions. **NDP C-386™** is a full implementation of AT&T's PCC with MS and ANSI extensions. **NDP Pascal-386™** is a full implementation of ANSI/IEEE Pascal, with extensions from C and Berkeley 4.2 Pascal.

NDP Fortran/C/Pascal-386/DOS each $595
NDP Fortran/C/Pascal-386/VM. $695
NDP Fortran/C/Pascal-386/UNIX. $795

Phar Lap 386/VMM extensions are supported by the VM releases of NDP Fortran, C, and Pascal, making it possible to compile and run programs as large as the free space on your hard disk.

Phar Lap Virtual Memory Manager . . $295
Phar Lap Development Tools $495

NDP Windows™ — NDP Windows includes 80 functions that let you create, store, and recall menus and windows. It works with NDP C-386 and drives all the popular graphics adapters . . . . . Library: $125, C Source: $250

NDP Plot™ — Calcomp compatible plot package that is callable from NDP Fortran. It includes drivers for popular plotters and printers. Works with CGA, MDA, EGA and VGA . . . $325

NDP/FFT™ — Includes 40 fast running, hand coded algorithms for single and double dimensioned FFTs which take advantage of the 32-bit addressing of the 386 or your hard disk. Callable from NDP Fortran with mW1167 and 80387 support $250
387FFT for 16-bit compilers $250

NDP to HALO '88 Graphics Interface — Enables you to call graphics routines in HALO '88 from NDP Fortran, C or Pascal. $100

Parallel Processing

Videoputer™ — The highest performance graphics card on the market. Uses a T800 and TI 34010 in conjunction with a 130 MHz Brook-Tree DAC. Includes one MB of system RAM, one meg of video RAM and a library of graphics primitives. Runs standalone or in conjunction with a transputer network and drives 32 and 64 KHz analog monitors $4995

Monoputer™ — The world's most popular PC transputer development product now extends the memory available for developing transputer applications from 2 to 16 MB. The board features a DMA bus interface for fast I/O.

Monoputer with T414 (0 MB) $995
Monoputer with T800 (0 MB) $1495

Quadputer™ — This board for the AT or 386 can be purchased with 2, 3 or 4 transputers and 1 or 4 MB of memory per transputer. Two or more Quadputers can be linked together to build networks of up to 100 or more transputers providing mainframe power . . . . . from $3495

Linkputer™ — Uses four Inmos C004 programmable cross bar switches. It allows the user to dynamically change the topology of the processors in a network. Using this board, it is possible to get 100% linkage among eight Quadputers and design larger, custom topologies. . . . . CALL

Transputer Compilers and Applications
These Parallel languages are designed for use with Monoputer2 and Quadputer2.

Logical Systems Parallel C $595
3L Parallel C or 3L Parallel Fortran . . $895
COSMOS/M - Finite element analysis . CALL
ParaSoft: Parallel Environment . . . $500
Performance Monitor . . . $500
C Source Level Debugger $500
T800/NAG™ (See NDP/NAG) $2750

387BASIC™ — Our 16-bit MS compatible compiler introduces numeric register variables to produce the fastest running 80x87 code on the market. $249

NUMERICS HOTLINE
(508) 746-7341

386 Your AT

386/387 Turbo™AT — This board plugs into your 80286 socket, allowing your IBM AT to run 32-bit protected-mode code written for the 80386. Includes an 80387 socket. The most cost-effective AT upgrade!

386/387 Turbo AT/16MHz $495
386/387 Turbo AT/25 MHz $695

Weitek-Based Coprocessor Boards

mW1167™ and **mW3167™** coprocessor boards are built at MicroWay using Weitek components. Each includes an 80387 socket.

mW1167-16 . . . . . $895
mW1167-20 . . . . . $1095
mW1167 Microchannel-16/20 . . . . . CALL
3167-20 . . . . . $995
3167-25 . . . . . $1295
3167-33 . . . . . $1695
mW3167/80387 Board $150

Intel Coprocessors and RAM

8087	$84	**8087-2**	$120
80287-8	$195	**80287-10**	$220
80387-16	$350	**80387-16SX**	$310
80387-20	$400	**80387-25**	$500

80C287A (CMOS) $280
287Turbo-12 (for AT compatibles) $350
RAMpak™ - one meg 32-bit memory module for Compaq 386 20/25 . . . . . $495
RAMpak™ - four meg $1995
256K 80ns DRAM $8.00
256K 100ns DRAM $6.50
256K SIMMs 100ns $99
1 meg SIMMs 100ns $300
(All of our Intel coprocessors include 87Test.)

Multi-User Accelerators

MicroWay's **AT8™** and **AT16™** intelligent serial controllers run 8 to 16 terminals under Unix or Xenix without bogging down your AT, 80386 or PS/2 PC. **AT8**: $895 **AT16**: $1295

12 MHz PC Accelerators

FastCACHE-286 12 MHz $299
SuperCACHE-286 12 MHz $399
FastCACHE-286 9 MHz $199

MicroWay

World Leader in PC Numerics

Corporate Headquarters: P.O. Box 79, Kingston, MA 02364 USA (508) 746-7341
32 High St., Kingston-Upon-Thames, U.K. 01-541-5466
USA FAX 508-746-4678 Germany 069-75-2023

Number **104** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

From C to Objects

by

Steve Sando

Maybe you're different, but I struggled to understand object-oriented programming (OOP). Most of the explanations involved things like car trunks and mailboxes. I have trouble accepting things when I don't know how they work. OOP was no different. But once I understood the mechanics, I saw the value in OOP and how to apply it.

In this article, we'll use some real programming examples to bridge the gap between traditional and object-oriented programming. We'll look at some programming problems whose solutions are quite close to the object-oriented method. Then we'll take the same problem set and convert it from straight C to an OOP extension for C, called SCC (see sidebar). By first pushing C as close as possible to OOP, the transition should be made simpler.

This article will also introduce the basics of the OOP vocabulary by applying OOP names to such standard C elements as structures and functions. Once you have a familiar frame of reference for concepts like "inheritance," you will be better prepared to make use of them.

The programming examples are familiar to all professional programmers. They are common problems involved with making reusable code, and they demonstrate the higher level of reusability achieved with object-oriented programming.

So, let's solve a problem using only traditional C programming techniques. We will solve it as best we can. There will be problems, seemingly unavoidable, when we are finished—problems that cause maintenance headaches and inelegant solutions.

OOP-Style Programming in Straight C—Roughly

Here's the problem: You have been assigned to convert some arrays of structures into two-way linked lists. The first step, of course, is to add a pair of pointers to the structures; **prev** to point to the previous item, and **next** to point to the next item. Then, you need to produce functions for adding and deleting members and printing the list. This is all fine for a linked list of only one type, but there are *many* types. In fact, the boss wants to be able to add new data types and still reuse your code. How do you avoid repeating the functions for each type of structure?

One way is to place the **prev** and **next** pointers in a fixed place in all the different structures. This way, the linked-list functions would not need to be concerned with the type of structure and would work only with the two pointers they know about.

Does this answer make you nervous? It means you have to pass different types of structures to the same function! Examples will demonstrate how this can be done. They assure the location of the **prev** and **next** pointers by embedding a special structure at the beginning of each data structure.

Take a close look at the examples. COOP.H (Listing 2) defines a set of structures that make use of a special structure called TWLL. COOP.C (Listing 3) has functions that assume they receive a pointer to TWLL. There are also specialized functions for each type of structure. To differentiate between the structure types, there is a type field. This allows the general-purpose functions (for example, **print_twll**) to identify the type.

SCC Language Specification

The SCC language is quite simple. You identify the class name, the name of the superclass, the data definitions, and method definitions. All code is done in straight C, using a special function called **Send** to send messages. The data definition is the same as any structure definition. Here is the grammar:

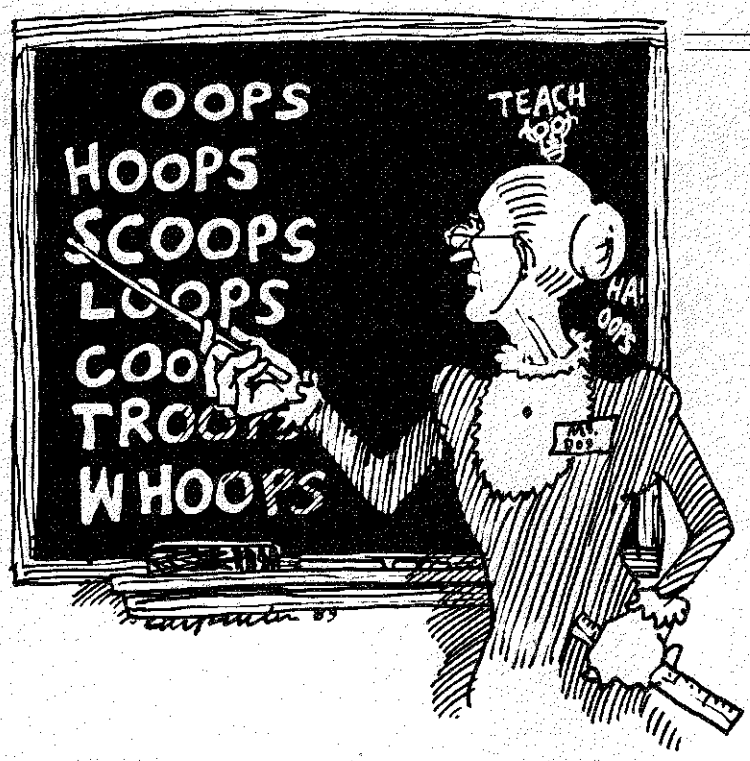
```
Class name-of-class : name-of-super-class
{ C style declarations } (optional)
Methods: (optional)
  message-name(parameters) { C code }
  message-name(parameters) { C code }
```

Comments begin with a pair of slashes (//) and continue to the end of the line. Everything between the curly braces ({ }) is standard C; therefore, comments inside must be in standard C format. Here is an example:

```
Class TwllHead:ObjRoot {
   Obj top;          /* comment */
   int count;
   }
   Methods:
       AddItem()         // comment
       { ... }
```

TwllHead is the class being defined, and ObjRoot is the superclass. "top" and "count" are two data items added to ObjRoot's data. "AddItem" is a message that TwllHead can receive.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Converting to Object-Oriented Programming Names

Structures as Classes

Rather than calling these "nested structures," call them "class definitions." The TWLL, DATA_TWLL, and MY_DATA_TWLL structures are each a class: TWLL is the root class, DATA_TWLL is a subclass of TWLL, and MY_DATA_TWLL is a subclass of DATA_TWLL. This is because MY_DATA_TWLL uses DATA_TWLL, which uses TWLL. We can also say that TWLL is a superclass of DATA_TWLL and DATA_TWLL is a superclass of MY_DATA_TWLL.

You might think that since DATA_TWLL contains a TWLL, it is the superclass. This is only natural, since DATA_TWLL has more information, but don't look at it that way. Think instead of TWLL as the parent and DATA_TWLL as "inheriting" the genes of TWLL and adding genes of its own.

Nesting as Inheritance

In OOP terms, this combining of structures to create more-advanced ones is called *inheritance*. Inheritance can be viewed as simply borrowing field definitions from another source. In our example code, the **structs** that use the TWLL functions must "inherit" the two pointer fields before they can make use of the link-list functions.

Let's summarize what we have in the code examples: a structure containing two pointers, a set of functions that operate on these pointers, and a set of structures that can make use of these functions by inheriting the two pointers (consistently placed at the beginning of the **struct**).

If you look closely at Figure 1, you'll notice that DATA_TWLL is inherited by MY_DATA_TWLL. This means that you can call a function that expects a DATA_TWLL with a MY_DATA_TWLL structure, since we were consistent in how we created the structures. The order was always the same: TWLL then DATA_TWLL, and finally the MY_DATA_TWLL information. Also, OTHER_DATA_TWLL inherits from DATA_TWLL. Therefore, the DATA_TWLL functions can be used by OTHER_DATA_TWLL structures, but MY_DATA_TWLL and OTHER_DATA_TWLL cannot share functions.

This method requires a lot of careful planning on the programmer's part. In an OOP system, this is made much simpler and always produces the necessary consistency.

Message Passing vs. Virtual Functions

There are two ways in which functions (methods) are tied to data types to allow you to override methods: with messages and with virtual functions. Messages are typically implemented internally as simple integers, which are stored in a table for each class. Each table entry has the message and a pointer to the function that handles the message for the associated class. When you send a message such as

```
Send(head,AddItem,x);
```

the **Send** function uses a pointer stored in the object "head" to find the class table. The message "AddItem" is then located in the table to find the right function, and control is passed to that function. There are many different methods used to look up the message. SCC uses a hash table to minimize the number of comparisons and to improve performance. In fact, message passing can provide better performance and take less code than the equivalent system using numerous switch statements (see Listing 3).

C++ uses virtual functions. Internally, when you first define a virtual function, C++ creates room in the structure (class definition) for a pointer to a table. The addresses for the virtual functions are kept in the table at offsets managed by the compiler. When you invoke a virtual function, the compiler generates code to use the pointer stored in the object, increments it based on the function name, grabs the function address from there, and calls it.

Using virtual functions is almost always more efficient than sending messages, but the difference is negligible. This slight performance improvement is not free: Every time you add a function to a base class, you must recompile all the child classes and any module that uses any of the classes. In addition, the C++ compiler I use seems to take as long as the C compiler, for a total of two times longer than straight C; so, minor changes can take quite a long time. In a message-passing system, only changes to the data definition require recompiling, and then only the subclasses that are affected. SCC accomplishes this by maintaining a global file that has all the class names and message names and numbers.

There is another advantage to message passing. With virtual functions, implementing the **Print** method (which has different versions for each subclass) is quite simple. However, to produce the same effect as the **SendAll** method, you need to add a **Print** method to the root class TWLL and pass a pointer to a virtual function to the **SendAll** function. This is not obvious in C++ and requires some strange constructs.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

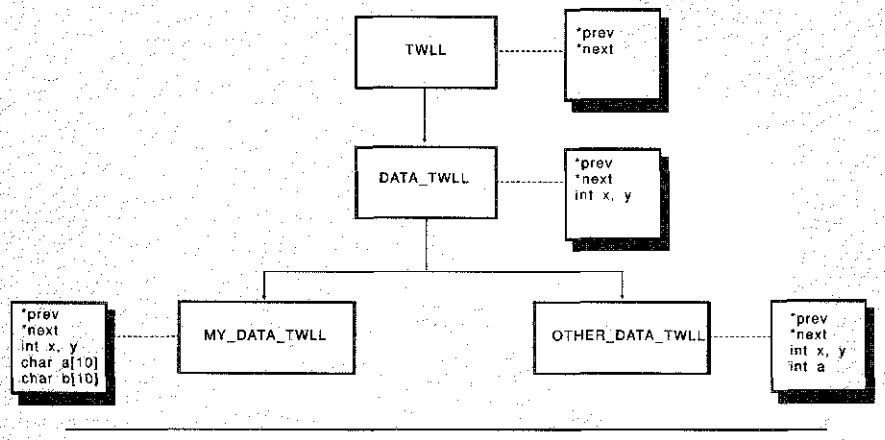


Figure 1. Hierarchy of structures used in TEST.C

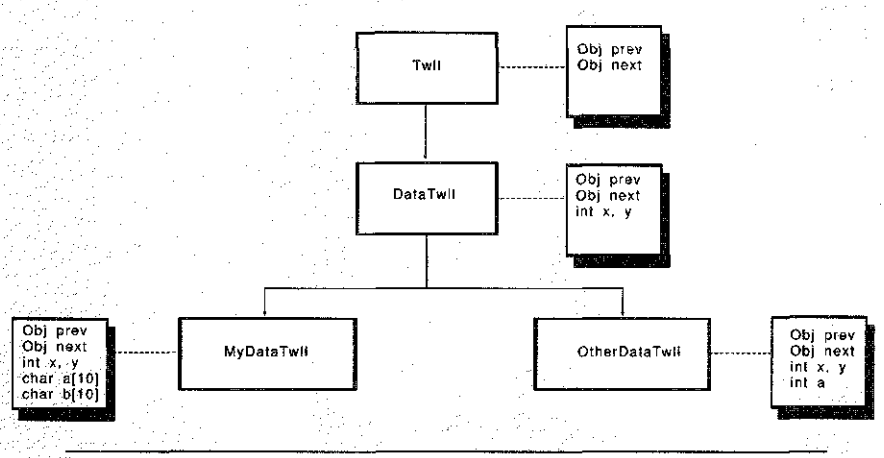


Figure 2. Hierarchy of classes used in TEST2.C

Functions as Methods

In reality, a class is more than the structure definition: It includes the functions that operate on the structure. While C does not "tie" together structures and functions, OOP languages always do. These functions that are linked to a structure are called *methods*.

For instance, the method **print_twll** is part of the TWLL class, because it operates on the TWLL structure. So, you can think of the structure DATA_TWLL and all the methods that work on this structure as a class.

A method always receives a pointer to the data type it operates on. Look at the functions in COOP.C (Listing 3). All of them receive a pointer as their first parameter. If you look at OOP.CLS (Listing 5), you will notice **self->** in all the methods, but **self** is never declared. Since methods always receive a pointer to the object, OOP compilers always add it for you. C++ uses the special identifier **this**, but the idea is the same. So, when you are coding a method, **self** is always a pointer to a structure that matches the class you are working on.

A class definition, therefore, has three parts: the identification of the superclass, its own data fields, and the list of methods that operate on the class. These functions are really a special form, and that is why we don't just call them functions. Look at the function **add_twll**. It operates on the TWLL structure. But is that really all? No, it can handle any structures that start with a previous or next pointer, for example, DATA_TWLL. These example functions and structures are brute force. Later, we will look more closely at how an OOP system ties functions and structures together with their superclasses to form a complete, correct class.

Messages vs. Function Calls

So, methods are just functions belonging to a structure definition. In TEST.C (Listing 1), the functions are called directly. However, in an OOP setup, methods are invoked by the sending of messages. Among other things, messages prevent the wrong function (that is, method) from being called. Another way to put it is that messages are used to invoke the method belonging to a piece of data.

TEST.C (Listing 1) and COOP.C (Listing 3) contain **switch** statements that check a type indicator so that the proper function will be called. This creates horrendous maintenance problems: Every time a new type is created, numerous **switch** statements need to be updated. Message passing avoids this problem, as we'll see later.

Also in COOP.C are four print functions. They all print, but there is one for each type of data. Isn't it more natural to have one function name called **Print** and define it differently for each data type? Look at OOP.CLS (Listing 5). There are still three print functions, but they are tied to their data types. Now compare TEST.C (Listing 1) and TEST2.C (Listing 4). In TEST.C, you have to determine the type and call the right function. In TEST2.C, you just send **Print** to all the members in the link list, and the system resolves which function to call.

If you think about it, functions *do* things. We naturally name them with verbs: read, write, open, close, and so on. More than that, functions do their things to the data that is passed. So, functions operate on data. In OOP, the focus is on data. Elements of data exist, and you want the data to do something, so you "send a message" and let the data determine how to accomplish it. You could sum it up by saying that traditionally, you ask a function to operate on some data, while with OOP, you ask the data to perform a function.

Objects

We've really come a long way in our analogy, and finally it's time for a definition: An *object* is a location in memory that contains data and that has been **malloc**ed to contain a

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

structure. This is a simple description, but one that is often overlooked.

In the following example, *abc* is the definition, or name, of the structure, and *def* is an occurrence of *abc*:

```
struct abc {int a, b;};
struct abc def;
```

That is, *abc* is the definition, or class, and *def* is the instance, or object.

So, what's the difference between an object and any other variable? Well, objects contain pointers to special processing information. This includes things like their size, the name of their class, and a list of messages and methods. The important point to keep in mind for now is that objects are simply areas in memory.

The code in COOP.C (Listing 3) does solve the problem of using a single set of functions to handle varying TWLLs, even with different structure types linked together. However, there are some problems. For one, adding new types means maintaining **switch** statements scattered throughout the program. For another, nested structures can lead to messy code, such as

```
item->inherit.inherit.prev
```

and allocating memory for structures can look like

```
xyz=(struct DATA_TWLL *)calloc(sizeof(struct
        DATA_TWLL),1);
```

If there were a way to identify superclasses without nesting, tying structures to their functions, and sending messages rather than calling functions, we would have it. That is exactly what the language extensions for object-oriented systems are for.

C Language Extensions: A Better Solution

There are several extensions to the C language that allow you to define classes, subclasses, messages, and methods. The samples provided in this article were created using SCC. The examples could also be expressed in much the same way using Objective-C, C_talk, or any of the now common object-oriented extensions to C. C++ also allows for defining classes and subclasses and provides an effect similar to message passing, called virtual functions (see sidebar).

Creating Objects

In any of the languages, creating objects is fairly consistent. Since objects are basically chunks of memory, they are created with **malloc** or some such function. Here is an example using SCC:

```
Obj    abc;
abc=New(TwllHead);
```

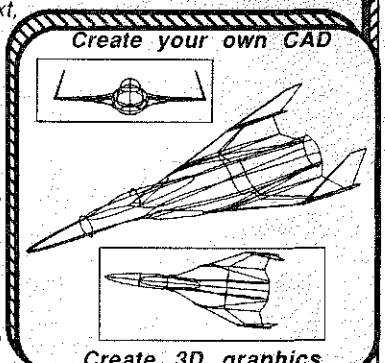
New is quite simple. Given the class name (which is actually a pointer to the class-definition information) it **calloc**s a block of memory the size of an object of that class type. It then sets a pointer in the beginning of the block to the class's dispatch table and returns a pointer to the block.

3 Dimensional Color Graphic Toolkit $175

CGL3D Limited offer

Create powerful 2D/3D graphic application programs on IBM or compatable PC computers with functions of an easy to use, high performance graphic library.

*2D/3D Point, Line, Text, Polygon, Fill, Surface
*World coordinates, multiple viewports
*True 3D perspective
*Text editor
*Element color/type
*Zoom, Transformation
*2D/3D Views
*2D/3D Rotation
*2D/3D Conversion
*PC MOUSE interface
*PLOT10 TCS interface
*Plot 3D surface data



*Easy to learn user manual, Source code for all samples
*Read 2D/3D data from file, display different views

The Library functions can be access by FORTRAN or C programs. Library is written in FORTRAN 77 and compiled with Microsoft FORTRAN compiler V4.

To order, Please send $175 plus $3 for shipping to:

F.G.SOFT,
P.O. BOX 2152,
HAWTHORNE,CA 90251-2152

Call today for a demo
213-542-5461

Number 307 on the Reader Service Card

Introducing C talk / Views™

Reduce C Programming for MS Windows by 75%

Introducing C_talk/Views™, a major productivity tool for Microsoft® Windows program development in C. Slash your MS Windows program development time by up to 75% through the powerfull object-oriented programming (OOP) technology of C_talk. With C_talk/Views, the most common elements of all MS Windows programs are packaged into reusable software components-object classes. Over forty software components are included as a complete applications development framework. These object classes provide an easy-to-learn and powerfully consistent interface to the rich functionality of MS Windows. C_talk/Views is also a real OOP environment. Use the software components as they are, or extend them to meet the most demanding applications.

Coming soon...C_talk/Views for the Apple Macintosh®... now your MS Windows program will port to the Mac with little or no change!

$450.00 In the US, add $5 for shipping. ($25 for Int'l handling)

CNS, Inc.
Software Products
7090 Shady Oak Rd.
Minneapolis, MN 55344
612-944-0170, Fax 612-944-0923

. . . providing and advancing object-oriented methodology.

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Obj is actually just a void pointer; it is used, though, to make the code more readable. Unless you know the class of **Obj,** you don't know the layout of the data. This is actually desirable—it is called *data hiding*, and all OOP systems provide it. The only way you can access the information in an object is by sending a message to it. This allows the designer of the class more control over how the data is accessed and how the data is represented. Both C++ and SCC provide ways to access the data directly—in C++ you specify the data as public, and in SCC you can cast **Obj**—but neither is recommended.

Sending Messages

Sending messages is where OOP programming takes off. It might seem at first that there is no advantage to sending a message over making a function call, but take your time. First look at how messages are sent, and then you'll see how a single message could invoke one of any number of functions depending on the class of the object. Look at OOP.CLS (Listing 5). Notice that there are three functions called **Print**. This is polymorphism—the same function with multiple implementations. When **main** sends **SendAll** with the message **Print** as a parameter, **SendAll** passes on the message **Print** to all members of the linked list. (The code for this is in OOP.CLS in class TwllHead, method **SendAll**). Since **Send** uses the object to locate **Print**, the right version of **Print** will always be called!

Let's look at this a little more closely. Examine the following code:

```
Obj    x, y;
x=New(MyDataTwll);
y=New(OtherDataTwll);
...
Send(x,Print);
Send(y,Print);
```

You already know about **Obj** and **New**. The issue here is **Send**. We want these two objects to print themselves. We don't care what they are; we just want to see their contents. So we send **Print**.

The two **Send** lines look like they do the same thing, and in a way they do. However, MyDataTwll and OtherDataTwll have each implemented **Print**. **Send** ensures that the right version of **Print** is invoked for both *x* and *y*. Without OOP, you would have to code numerous **if-or switch** statements to determine types.

This ability to change the flow of control based on data is called *dynamic*, or *late, binding*. If you are used to knowing the exact flow of control in your programs, you'll have to take a deep breath and relax. In conventional programming, you knew facts such as **A** calls **B** and **B** calls **C**. In OOP programming, all you know for sure is that the method **A** for class X calls **B**. You don't know whose version of **B** will get called, because that is determined by the data and you don't know what type of data you were passed.

You might think everything would get out of hand very quickly. It doesn't. It is not long before you learn to focus your attention on specific types of data and specific methods and let the system worry about which calls what when.

The Fastest Fortran 77/386 Compiler Around!!!

Proven compilers get solid results

- Compiles 13,000 lpm on a 386
- Runs under MS-DOS with DOS extenders
- Optional checkout mode with run-time diagnostics
- Full window-based source level debugger
- Dynamically allows linking to libraries during program's execution
- Contains a full 32-bit in-line symbolic assembler with window-based debugger
- Addresses up to 4 Gigabytes of memory

OTG™ SYSTEMS, INC.

Distributed by OTG Systems, Inc.

Our compilers run faster, do more, cost less and increase system and people productivity!

P. O. Box 239, Clifford, Pennsylvania 18413-0239
(717) 222-9100 Fax: (717) 222-9103

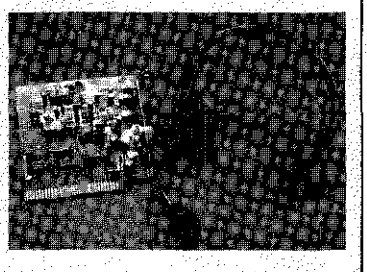
MS-DOS is a trademark of Microsoft Corporation...FTN77/386 is developed by the UNIVERSITY OF SALFORD

Number **270** on the Reader Service Card

VOICE MASTER KEY
VOICE RECOGNITION SYSTEM
FOR PC/COMPATIBLES & TANDY 1000 SERIES
A FULL FEATURED VOICE I/O SYSTEM

GIVE A NEW DIMENSION TO PERSONAL COMPUTING. . .The amazing **Voice Master Key System** adds recognition to just about any program or application. Voice command up to 256 keyboard macros from within CAD, desktop publishing, word processing, spread sheet, or game programs. Fully TSR and occupies less than 64K. Instant response time and high recognition accuracy. Voice recognition tool-box utilities are included. **A genuine productivity enhancer!**

SPEECH RECORDING SOFTWARE. . .Digitally record your own speech, sound, or music to put into your own software programs. Software provides sampling rate variations, graphics-based editing, and data comoression utilities. Create software sound files you can add to macros for voice recognition verification response. **a complete, superior speech and sound development tool.**



SOFTWARE CONVERSION CODES. . .The **Voice Master Key System** operates a growing list of third party talking software titles using synthesized phonetics (text - to - speech) or digitized PCM, ADPCM, and CVSDM encoded sound files. **Voice Master Key system does it all!**

EVERYTHING INCLUDED. . .**Voice Master Key System** consists of a plug-in card, durable lightweight microphone headset, software, and manual. Card fits any available slot. External ports consist of mic inputs and volume controlled output sockets. **High quality throughout, easy and fun to use.**

ONLY $149.95 COMPLETE

ONLY $89.95 FOR TANDY 1000 SL/TL MODELS SOFTWARE ONLY—REQUIRES TANDY BRAND ELECTRET MICROPHONE.

ORDER HOTLINE: (503) 342-1271 Monday-Friday, 8AM to 5 PM Pacific Time

Visa/MasterCard, company checks, money orders, CODs (with prior approval) accepted. Personal checks subject to 3 week shipping delay. Specify computer type and disk format (3½" or 5¼") when ordering. Add $5 shipping charge for delivery in USA and Canada. Foreign inquiries contact Covox for C & F quotes. 30 DAY MONEY BACK GUARANTEE IF NOT COMPLETELY SATISFIED. ONE YEAR WARRANTY ON HARDWARE.

CALL OR WRITE FOR **FREE** PRODUCT CATALOG

COVOX INC.
675 Conger Street, Eugene, OR 97402
TEL: 503-342-1271 • FAX: 503-342-1283

Number **277** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Overriding Methods

In OOP.CLS (Listing 5), you saw that the message **Print** was defined in three different classes: DataTwll, MyDataTwll, and OtherDataTwll. Since MyDataTwll is a subclass of DataTwll, the **Print** message in MyDataTwll is an *override;* in other words, we told the SCC compiler that MyDataTwll has its own version of **Print**. So, when the message **Print** is sent to a MyDataTwll object, the function **MyDataTwll_Print** is called, instead of **DataTwll_Print**.

This is a very important attribute of OOP programming. So, let's see how it works, using the following code:

```
Obj    abc, def;
abc=New(DataTwll);
def=New(MyDataTwll);
Send(abc,Print);
Send(def,Print);
```

Send gets *abc* first. It finds the dispatch table for DataTwll (the class of object *abc*) and discovers that the function for **Print** is in class DataTwll. Control is passed. Internally, the function is named **DataTwll_Print**. This avoids problems, since LINK can't handle three functions all called **Print**.

Send then gets *def*. It finds the dispatch table for MyDataTwll and discovers that the function for its **Print** is **MyDataTwll_Print**. Control is passed.

Now, what if we say:

```
Send(def,SetX);
```

Message **SetX** is not defined for MyDataTwll. But it *is* defined for its superclass, DataTwll. This time, **Send** passes control to **DataTwll_SetX**. This is inheritance: Just as MyDataTwll inherits the data definition from DataTwll, it also inherits the methods. Therefore, we say that a subclass can do anything that its parent can do.

Passing Messages on to the Superclass

In the above example, we saw how the message **Print** was dispatched to the right function. Now, sometimes we don't want to override the original method; we only want to supplement it. For example, **DataTwll_Print** prints something that we wish for MyDataTwlls. We could repeat the code, but there is a better way.

The perfect solution is to pass control to the superclass's **Print** method. This is done with the **SendSuper** function. Look at **MyDataTwll_Print**. After it prints its part, it calls **SendSuper**. This passes the message "up the chain." Now, **DataTwll_Print** gets the opportunity to do its thing.

Style Differences

Object-oriented programming is a combination of programming language tools and style. In general, conventional programs are organized by the code; that is, you tend to think in terms of the **main** module or function, its subfunctions, their subfunctions, and so on. The organization of data and the relationships between differing types are "hard coded" and used as needed.

MISOSYS

Programmer's Journal downloads are now available from MISOSYS' forum on Compuserve: GO PCS49

Are you still fussing with floppies for BACKUP? Our QIC40 JUMBO tape drive from MISOSYS is your solution!

The Colorado Memory Systems' JUMBO tape drive is one drive that fits all computers. It comes ready for internal use in AT's, XT's, and PC's, and connects to your floppy disk controller. Our adaptor board can be used to connect JUMBO when all FDC connectors are in use. Kits are available for XT/AT or PS/2 machines which convert Jumbo to external use.

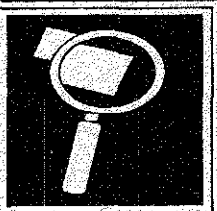
- In about 5.5 minutes, JUMBO backs up 10MB's file-by-file - the fastest in the industry! 40MB's gets backed up in about 18 to 20 minutes. Uses industry-standard DC2000 tape cartridges.
- JUMBO plugs into your floppy disk controller to save cost, power, and a slot. Needs 5-1/4" (or 3.5" with faceplate) mounting slot.
- Our tape adapter board mounts in your host computer to provide an additional tape port — especially useful in 286 and 386 computers. When used with our external JUMBO, it lets you share your drive between computers. Note: external adaptor includes "Tape Adaptor"
- JUMBO has custom chips, high speed brushless motors, automatic circuits, no pots to vibrate out of calibration, and fewest parts to assure long life and technical leadership for years to come.

DJ10 Jumbo	$295 ($5S&H)
Tape Adaptor	$75 ($3S&H)
External Adaptor	$110 ($5S&H)
DC2000 tape	$22.50

M/C & VISA. S&H are U.S. only.

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS
or 703-450-4181

Now get the clearest view of your disk drive and memory contents with our DED86™



Powerful features in Version 2!

When you need to travel through your disk drive, why settle for a tool that isolates you from the road? DED86 gives you the direct controls you need to explore your disk. It's a full-screen sector-oriented disk/file editor and a page-oriented memory editor. When you want to "unerase" erased files, DED86's flexible KEEP facility does the job without you fussing over FATs.

Check out this list of features

- ✓ Look by cylinder/head/sector, sector or cluster; even reserved/hidden sectors
- ✓ Scan free clusters to search for erased data
- ✓ Jump about subdirectories
- ✓ Keep sectors & clusters for writing to a file; Great for partial file recovery
- ✓ Edit bytes in hexadecimal or ASCII, zap in 0s; Undo last edit!
- ✓ Search your disk drive, a file, or memory for ASCII or hexadecimal strings
- ✓ Touch a directory file entry with your date and/or time
- ✓ Obtain complete disk statistics in one screen; includes usage data
- ✓ Alter file attributes: archive, system, hidden, read
- ✓ Save/Restore sectors to/from auxiliary buffers; for moving data around
- ✓ DOS subshell available while using DED86 lets you shell to other programs
- ✓ Handles 5.25" & 3.5" drives & RAM disks; even large partition drives

DED86 reduced by more than 50%; Now only $29.95 + $2S&H

Fujitsu Floppy Disk Drives:	5.25" 360K 1/2 height	$75+$4S&H
	3.5" 720K in 5.25" frame	$85+$3S&H
	3.5" 1.44M in 5.25" frame	$99+$3S&H

Number 121 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

When using object-oriented tools, you think in terms of data. There are base types of data, data types derived from the base types, types derived from these, and so on. Also, the code for each data type is tied to the data definition.

The difference can almost be summarized as data used by code (conventional) versus code used by data (OOP). This results in a fundamentally different organization of a program and a different way of looking at your programming problems.

Program execution is not quite the same, either. In an object-oriented application, the program will usually start life by creating a single object. This object is then "sent a message"—something like "Go." This object will then create other objects, which in turn create more objects. Objects are constantly being created, sending messages to one another, and dying. As the application winds down, more and more objects complete their tasks and die. Finally, the first object sends itself "Kill" (or some similar message), dies, and returns control to **main**.

Object-oriented programming is much more than just style. Without language extensions of some kind, it is almost impossible to manage your code. The only real way to get a feel for OOP is to try it. Pick up a copy of SCC, C++, or one of the many other fine products available today, and jump in. PJ

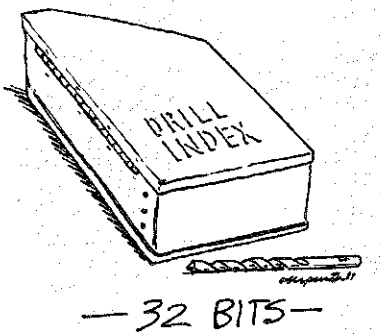
Steve Sando is currently working with Application Development Systems. He has been designing and writing commercial software products for IBM micros and mainframes for the past ten years. You can contact Steve via CompuServe, user ID 76357,1327.

Code follows.

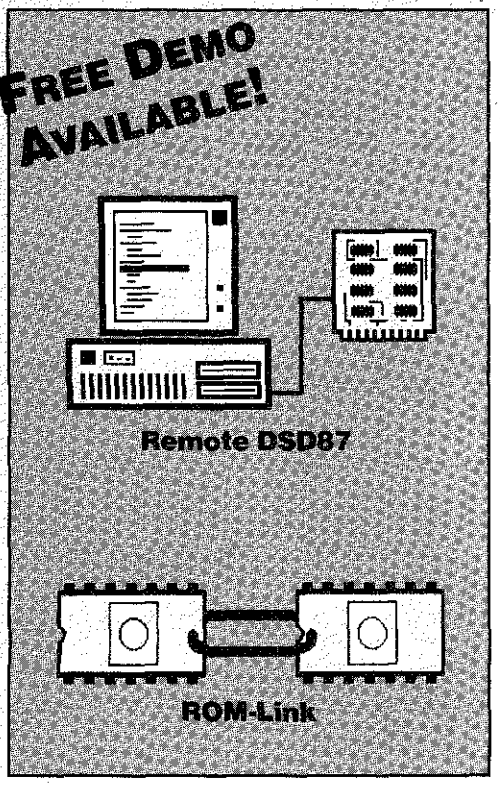
SCC is a trademark of Steve Sando. SCC does not modify your source but does add classes, inheritance, message passing, and object management to C. This means it stays compatible with standard C code, libraries, debuggers, and so on. Documentation and a disk containing the SCC compiler, sample programs, run-time source, and library source for classes for binary trees, linked lists, stacks, variable-length arrays, hash tables, and so on, is available from the author. Send a check for $25 for handling to:

Steve Sando
1700 Don Ave.
San Jose, CA 95124

Objective-C is a registered trademark of The Stepstone Corporation. C_talk is a trademark of CNS, Inc.



For the Ultimate in Embedded Systems Programming.



Debug Debug Debug Debug Debug

Remote DSD87 is the best source debugger for embedded development and the only product offering high performance and advanced software In Circuit Emulator capabilities. This package and your compiler form a complete solution for the development of embedded programs. Order now and receive a FREE copy of Native DSD87, a $125 value.

Remote DSD87 (supports both Microsoft and Turbo C) . . . $395
Remote DSD87 OMF (includes Intel OMF support) $795

Link Link Link Link Link Link Link Link

ROM-Link is the first integrated link and locate utility to directly generate COM, EXE, SYS, HEX, or Intel OMF files. The only linker ever guaranteed faster than any present or future competitors. Supports Microsoft debugging information. Receive a FREE librarian and cross reference utility with your order!

ROM-Link $395

Soft Advances
10811 Washington Bl., Ste. 205 • Culver City, CA 90232 • (213) 559-7015

Number **33** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800-234-0386.
Mail-order information on pages 8 and 88.

Code from "From C to Objects," by Steve Sando

1

LISTING 1

```
/* TEST.C - Test the twll functions */
#include "stdio.h"
#include "malloc.h"
#include "coop.h"

/* This is the TWLL header that contains the top of the list pointer */

struct TWLL_HEAD header={0};

main()
{
    int      x;
    struct DATA_TWLL        *twll_item;
    struct MY_DATA_TWLL     *my_twll_item;
    struct OTHER_DATA_TWLL  *other_twll_item;

    /* add 3 entries of each type to the list */
    for (x=3;x;x--) {
        twll_item=                          /* create a new DATA_TWLL */
            (struct DATA_TWLL *)
            calloc(sizeof(struct DATA_TWLL),1);

        add_data_twll(&header,twll_item,DATA);
        twll_item->x=x;

        my_twll_item=               /* create a new MY_DATA_TWLL */
             (struct MY_DATA_TWLL *)
            calloc(sizeof(struct MY_DATA_TWLL),1);

        add_data_twll(&header,my_twll_item,MY_DATA);
        strcpy(my_twll_item->a,"Sample.");

        other_twll_item=                /* create a new OTHER_DATA_TWLL */
             (struct OTHER_DATA_TWLL *)
            calloc(sizeof(struct OTHER_DATA_TWLL),1);

        add_data_twll(&header,other_twll_item,OTHER_DATA);
        other_twll_item->a=x+100;
        }

    /* print the contents of the list */
    printf("printing lines in current list \n");
    for(twll_item=header.top;
        twll_item;
        twll_item=twll_item->inherit.next) {

        switch (twll_item->inherit.type) {
            case DATA:
                print_data_twll(twll_item);
                break;
            case MY_DATA:
                print_my_data_twll(twll_item);
                break;
            case OTHER_DATA:
                print_other_data_twll(twll_item);
                break;
            }
        }

    exit(0);
}
```

2

LISTING 2

```
/* COOP.H - contains structures for simulated Objects */
// -----------Structure TWLL-----------------------------------------
//  This is our root structure or Super Class.
struct TWLL {
    struct TWLL *prev;
    struct TWLL *next;
    int     type;
#define DATA    1
#define MY_DATA     2
#define OTHER_DATA 3
  };

// -----------Structure TWLL_HEAD------------------------------------
//  This is our header struct for the linked list
struct TWLL_HEAD {
    struct TWLL *top;
    int   cnt;
    };

// -----------Structure DATA_TWLL------------------------------------
//  Here is a struct that inherits two pointers from TWLL.
//  By always placing TWLL at the beginning then
//  TWLL functions can make the assumption that both pointers
//  are at the start of the structure.
struct DATA_TWLL {
    struct TWLL inherit;
    int x;
    int y;
    };

// -----------Structure MY_DATA_TWLL---------------------------------
//  Here is a stuct that inherits the two
//  pointers from TWLL and the x and y fields from DATA_TWLL.
struct MY_DATA_TWLL {
    struct DATA_TWLL inherit;
    char   a[10];
    char   b[10];
    };

// -----------Structure OTHER_DATA_TWLL------------------------------
//  Here is one last structure using TWLL and DATA_TWLL.
//  Since it uses the same structure (DATA_TWLL) as MY_DATA_TWLL,
//  this class is a sibling of MY_DATA_TWLL
struct OTHER_DATA_TWLL {
    struct DATA_TWLL inherit;
    int a;
  };
```

LISTING 3

```
/* COOP.C - functions to handle twll structure and related structures */
    add_twll()              - insert a new item on the top of the list
    add_data_twll()         - add a new data_twll item by calling add_twll()
    print_twll()            - print a description of the item's type
    print_data_twll()       - print values and call print_twll()
    print_my_data_twll()    - print values and call print_data_twll()
    print_other_data_twll() - print values and call print_data_twll()
*/
#include "coop.h"
/* Add a new item to the top of a two-way linked list */
add_twll(head,twll,type)
struct TWLL_HEAD *head;
struct TWLL *twll;
int             type;
{
    struct TWLL *cur;

    cur         =head->top;
```

Code continues . . .

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Sandō code, continued

3

```
    twll->prev =0;
    twll->next =cur;
    if (cur) {
        cur->prev  =twll;
        head->top  =twll;
        head->cnt++;
        }
    head->top=twll;
    twll->type=type;
}
/* This function inits any struct that starts with DATA_TWLL.
 These structures must have x and y inited to 5 and 6.
 It then calls the regular add function to insert the struct.
*/
add_data_twll(head,twll,type)
struct HEAD_TWLL *head;
struct DATA_TWLL *twll;
int              type;
{
    twll->x=5;
    twll->y=6;
    add_twll(head,twll,type);
}
/* Print the data type */
print_twll(twll)
struct TWLL *twll;
{
    switch (twll->type) {
        case DATA:
            printf("type is DATA\n");
            break;
        case MY_DATA:
            printf("type is MY_DATA\n");
            break;
        case OTHER_DATA:
            printf("type is OTHER_DATA\n");
            break;
        }
    return;
}
/* Print the data in a DATA_TWLL */
print_data_twll(twll)
struct DATA_TWLL *twll;
{
    printf("x is %d, y is %d \t",twll->x,twll->y);
    print_twll(twll);
}
/* Print the data in a MY_DATA_TWLL */
print_my_data_twll(twll)
struct MY_DATA_TWLL *twll;
{
    printf("a is %s \t",twll->a);
    print_data_twll(twll);
}
/* Print the data in an OTHER_DATA_TWLL */
print_other_data_twll(twll)
struct OTHER_DATA_TWLL *twll;
{
     printf("a is %d \t",twll->a);
     print_data_twll(twll);
}
```

LISTING 4

```
/* TEST2.C - Same as TEST.C, except using objects */
#include "stdio.h"
#include "pubclass.h"

main()
```

4

```
{
    int x;
    Obj item;
    Obj header=New(TwllHead);

    /* add 3 entries of each type to the list*/
    for (x=3;x;x--) {
        item=New(DataTwll);         /* create a new DataTwll Object */
        Send(header,AddItem,item);
        Send(item,SetX,x);

        item=New(MyDataTwll);       /* create a MyDataTwll */
        Send(header,AddItem,item);
        Send(item,SetA,"Sample.");

        item=New(OtherDataTwll);    /* create an OtherDataTwll */
        Send(header,AddItem,item);
        Send(item,SetA,x+100);
        }

    /* print the contents of the list */
    Send(header,SendAll,Print);

    exit(0);
}
```

LISTING 5

```
//    OOP.CLS - Methods for two way linked lists
Include "Root.cls"

// ------------Class TwllHead---------------------------------------

Class TwllHead : ObjRoot{
    Obj     top;
    int     cnt;
    }
Methods:

// ----------------------------------------------------------------
// Add a new twll item to a list
AddItem(Twll_s *twll)
    {
        struct Twll_s *cur;

        cur        =self->top;
        twll->prev =0;
        twll->next =cur;
        if (cur) {
            cur->prev  =twll;
            self->top  =twll;
            self->cnt++;
            }
        self->top=twll;
        return;
    }

// ----------------------------------------------------------------
// This method passes a message on to all the Objects in the list.
SendAll(int pass_msg)
    {
        Twll_s      *x;

        for(x=self->top; x ;x=x->next)
            Send(x,pass_msg);
        return;
    }

// ------------Class Twll-------------------------------------------
```

Code continues . . .

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED.

C CODE FOR THE PC

source code, of course

	MS-DOS File Compatibility Package (create, read, & write MS-DOS file systems on non-MS-DOS computers)	$500
	Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
NEW!	C+O Data Structures Library (multiple inheritance, messaging, excpetion handling, call-back)	$375
NEW!	dB2c (dBase-to-C translator; includes dB_Files for C anddB_Tools for C)	$325
NEW!	pBase (relational DBMS with debugging calls and sparse table & repeated field support)	$325
	CQL Query System (SQL retrievals on B-trees plus windows)	$325
Updated!	GraphiC 5.0 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$290
NEW!	Code Base (database manager, dBase and Clipper compatibile indes & data files; Version 4.0)	$260
NEW!	MEWEL (extensible window and event library by Magma Software; message-passing & object-oriented; SAA-compatible; dialog editor)	$250
Updated!	TurboTEX (Release 2.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
	Greenleaf Data Windows (windows, menus, data entry, interactive form design; specify compiler)	$220
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF; specify compiler)	$175
NEW!	QuickGeometry Library (large collection of mathematics, graphics, display & DXF subroutines for CAD/CAM/CAE/CNC)	$170
	Sherlock (C debugging aid)	$170
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$165
	AT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for ATs)	$160
	Greenleaf Functions (296 useful C functions, all DOS services; specify compiler)	$160
	WKS Library Version 2.0 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	OS/88 (U**x-like operating system, many tools, cross-development from MS-DOS)	$150
	ME Version 2.1 (programmer's editor with C-like macro language by Magma Software; Version 1.31 still $75)	$140
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
	TurboGeometry (library of routines for computational geometry)	$125
Updated!	Install 2.3 (automatic installation program; user-selected partial installation; CRC checking)	$120
	TE Editor Developer's Kit (*full screen editor, undo command, multiple windows*)	$105
Updated!	Minix Operating System (Version 1.3; U**x-like operating system, includes manual)	$105
	HyperText Viewer (simple hypertext system; multi-file documents; includes Tiny Curses)	$100
Updated!	PC/IP (CMU/MIT TCP/IP for PCs; Ethernet, Appletalk & NETBIOS drivers, RVD, update by Dan Lanciani)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
	The Profiler (program execution profile tool)	$100
	QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
NEW!	Otter 1.0 (beautiful theorem-prover by Bill McCune; includes manual & two books by Wos; complete starter kit)	$80
	C Windows Toolkit (pop-up, pull-down, spreadsheet, CGA/EGA/Hercules)	$80
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
	Polyglot Lisp-to-C Translator (includes Lisp interpreter, Prolog, and simple calculus prover)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
Updated!	Make (macros, all languages, built-in rules)	$75
	evalO (C function to evaluate ASCII infix expression string; 17 built-in functions)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	Professional C Windows (lean & mean window and keyboard handler)	$70
NEW!	ScreenLib (simple screen definitions, windows, pop-up menus, context-sensitive help)	$65
	Heap Expander (use LIM-standard expanded memory as an extension of the heap)	$65
	lp (flexible printer driver; most popular printers supported)	$65
	SYSKIT (rommable or TSR debug/monitor; easily expanded)	$60
	Quincy (interactive C interpreter)	$60
	Symtab (general-purpose symbol table construction and management package)	$60
	PTree (general-purpose parse tree construction and management package)	$60
Updated!	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
NEW!	Async-Termio (Unix-compatible general terminal interface for MS-DOS	$55
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	Polyglot TSR Package (includes reminder, bookmark, virus catcher, cache manager, & speech generator)	$50
	Multi-User BBS (chat, mail, menus, sysop displays; does not include modem driver)	$50
NEW!	Fortran-to-C Translator by Polyglot	$40
	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$40
	Heap I/O (treat all or part of a disk file as heap storage)	$40
	Biggerstaff's System Tools (multi-tasking window manager kit)	$40
	OOPS (collection of handy C++ classes by Keith Gorlen of NIH; Version 2.2)	$35
	Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor; now includes documentation)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
	CLIPS (rule-based expert system generator, Version 4.2)	$35
	Tiny Curses (Berkeley curses package)	$35
	Polyglot RAM Disk (change disk size on the fly; includes utilities)	$30
	Clisp (Lisp interpreter with extensive internals documentation)	$30
	Translate Rules to C (YACC-like function generator for rule-based systems)	$30
	6-Pack of Editors (six public domain editors for use, study & hacking)	$30
	Crunch Pack (14 file compression & expansion programs)	$30
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Wirth standard Pascal)	$25
	ICON (string and list processing language, Version 7.5)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX; Version 1.1)	$25
	LEX (lexical analyzer generator; an oldie but a goodie)	$25
	AutoTrace (program tracer and memory trasher catcher)	$25
	Data Handling Utilities in C (data entry, validation & display; specify Turbo C or Microsoft)	$25
	Arrays for C (macro package to ease handling of arrays)	$25
	A68 (68000 cross-assembler)	$20
	List-Pac (C functions for lists, stacks, and queues)	$20
	XLT Macro Processor (general purpose text translator)	$20
	Data	
	Protein Sequences (over 10,000 sequences; includes demo disk of Pearson FAST/A programs)	$60
	Smithsonian Astronomical Observatory Subset (right ascension, declination, & magnitude of 258,997 stars)	$60
	Moby Words (500,000 words & phrases, 9,000 stars, 15,000 names)	$55
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
	USNO Interactive Computer Ephemeris (high-precision moon, sun, planet & star positions)	$30
	NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
	U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

acw!info@uunet.uu.net

Voice: (512) 258-0785
BBS: (512) 258-8831
FAX: (512) 258-1342

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

Number 30 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Sando code, continued

5

```
//  Abstract Root class for all object that can be added to a list
Class Twll : ObjRoot {
    Obj     prev;
    Obj     next;
    }

// ------------Class DataTwll-----------------------------------------
Class DataTwll : Twll {
    int     x, y;
    }

Methods:

// ------------------------------------------------------------------
// Initialize a DataTwll object for x and y
int ObjInit()
    {       // init x and y
        self->x=5;
        self->y=6;
        SendSuper(SuperDataTwll,msg);           /* pass the init msg up */
        return;
    }

// ------------------------------------------------------------------
// Set the value of X
SetX(int x)
    {
        self->x=x;
    }

// ------------------------------------------------------------------
// Print the contents of a data TWLL
Print()
    {
        printf("x is %d, y is %d \t",self->x,self->y);
        Send(self,ObjPrintName);
    }

// ------------Class MyDataTwll---------------------------------------
Class MyDataTwll : DataTwll {
    char    a[10], b[10];
    }

Methods:

// ------------------------------------------------------------------
// Print the contents of a MyDataTwll
Print()
    {
        printf("a is %s \t",self->a);
        SendSuper(SuperMyDataTwll,msg);
    }

// ------------------------------------------------------------------
// Set the value of a
SetA(char *str) { strcpy(self->a,str); }

// ------------Class OtherDataTwll------------------------------------
Class OtherDataTwll : DataTwll {
    int     a;
    }

Methods:

// ------------------------------------------------------------------
// Print the contents of an OtherDataTwll
Print()
    {
        printf("a is %d \t",self->a);
```

6

```
        SendSuper(SuperMyDataTwll,msg);
    }

// ------------------------------------------------------------------
// Set the value of a
SetA(int a)
    {
        self->a=a;
    }

// ------------------------------------------------------------------
OtherCode {
abort_msg(char *msg)
{
    puts(msg);
    exit(1);
}

}
```

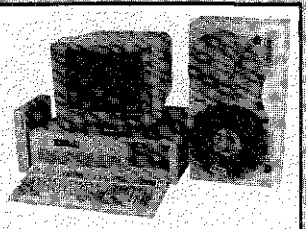
LISTING 6

```
typedef int (*funcp)();

typedef struct METH_T {
    int    msg;                 // msg on top to improve compare time
    funcp  procc;               // pointer to method for msg
    struct METH_T near *next;   // next entry in this hash chain
    } METH_TABLE;               // entry in method table

#define MAX_METHS 32            // must be multi of 2
typedef struct class_def {
    METH_TABLE near *tbl;       // pointer to method table for this class
    char       *name;           // asciiz string of class name
    int        size;            // size of objects in this class
    struct class_def  *super;
    METH_TABLE near *hash[MAX_METHS]; // hash table for method look-ups
    } CLASS_DEF;

typedef struct {
    CLASS_DEF near *class;
    } *Obj_x;

/*-------------------------------------------------*/
void *send(obj,msg)
Obj_x obj;
int msg;
{
    METH_TABLE  near *tbl;
    CLASS_DEF   near *clas;
    int hash;

    hash=msg & (MAX_METHS-1);

    clas=obj->class;
    if ( !(tbl=clas->hash[hash]) ) {
        send_org(clas);                     // initialize the hash table
        tbl=clas->hash[hash];
        }

    while (tbl->msg != msg)                 // search hash chain for msg
        tbl=tbl->next;

    (*tbl->procc)();                        // NOTE:
                                            //  need to convert this to a jump
    return;
}
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Desktop 9-Track Tape Subsystem

Now, 9-track tape lets your micro exchange data with minis and mainframes



Simply exchange data files on a reel of 9-track tape.

9-TRACK is the first choice for file interchange among data processing professionals. Now, Qualstar's low cost 1/2-inch 9-track Streaming tape systems bring full ANSI data interchange to IBM PCs or Macintosh, giving your micro the freedom to exchange data files with nearly any mainframe or minicomputer in the world.

Available in both 7" and 10-1/2" versions, compact Qualstar tape drives can sit on your desktop, *using less space than an ordinary sheet of paper.* Systems include DOS or XENIX compatible software, coupler card and cables. High reliability 1600 or 6250 BPI capability may be used for disk backup as well as data interchange. Discover the big advantage 9-track tape has over other micro/mainframe links.

DEC HP ALTOS NCR — IBM MAINFRAMES — AT&T UNISYS APOLLO SUN

#1 Selling 9-Track Systems on the Desktop

QUALSTAR

Call us today!

FOR DETAILS AND TO ORDER
FAX (818) 882-4091
PHONE (818) 882-5822

9621 Irondale Ave., Chatsworth, CA 91311

© 1989 Qualstar Corp. All product and company names and trademarks are the exclusive property of their respective owners.

Number **97** on the Reader Service Card

EVERLOCK

COPY PROTECTION

Designed for clone compatibility & strength

- Thwarts ALL Bit-copy Software
- Protect any COM/EXE w/o Source changes
- Shut down Debug Tracing & Disassemblers
- Supports all Hard & floppy Disk formats
- Restrict or allow Network Installs
- Limit program use by Date or #Executes
- Remotely reset Installs, Date or #Executes
- No damaged media or I/O plugs

EVERTRAK

SECURITY

If you HATE Copy Protection, but still want to protect your program from hackers and reverse engineering, **EVERTRAK** allows you to:

- Protect any COM/EXE w/o Source changes
- Shut down DEBUG systems and disassemblers
- Embed/display hacker-proof Customer information
- Restrict use by date-range and/or media-type
- Track unauthorized copies back to their source
- Totally invisible to End Users

For IBM & Compatibles. 30 day money back guarantee.
Free information and demonstration disk available.

TeleCheck VISA MasterCard

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085

(800) 227-0644 • (816) 776-2700 • FAX (816) 776-8398

Number **51** on the Reader Service Card

NEW BBS

SCREEN MANAGER P·R·O·F·E·S·S·I·O·N·A·L

ADVANCED WINDOWING

• The Screen Manager Professional gives your applications the finest bullet proof windowing interface available. Although other products may boast of hundreds of functions, they often involve needless repetition. Our functions are elegant and powerful. Each accomplishes a variety of tasks with one command. All window functions are written in hand-optimized assembly language to provide the fastest possible response and smallest possible size.

ORDER

- 30 day Money Back Guarantee
- Screen Manager Professional $140
- Source Code $140
- Sampler Library ..... $ 40

• Specify Compiler when ordering • Include UPS shipping: ground $5, overnight $20, 2nd day air $10, outside U.S. $30, COD $5. • Visa, Mastercard, check or money order. • Payment must be in U.S. dollars drawn on a U.S. bank. • PA residents add 6% sales tax.

POWERFUL

- Amazing Speed
- Incredible Menus
- Context Sensitive Help Editor
- OS and Compiler Independent
- Window Functions are 100% Assembly
- Keyboard Filtering for Data Entry Systems
- Full Video Adapter Support
- Compact Linkable Library
- Background Processing
- Customizable Memory Allocation Function

VERSIONS AVAILABLE FOR

•Microsoft C •Zortech C++
•Quick C •Turbo C •QuickBasic
•Turbo Pascal (available 7/89)

AMAZING

"...an amazingly complete collection of window commands." The support functions are easy to use and very powerful. This library warrants serious consideration."
Computer Language-May '89

• Free yourself to concentrate on the substance of your program. Over 120 functions provide all the windowing tools needed to develop serious applications whether you are a professional or a hobbyist.

• The Screen Manager Professional can help you create easy to use, visually stimulating, programs with minimum effort. Now powerful windowing is available to you at an affordable price.

NEW BBS

Demos, Software, Tech Support- A forum for the Professional Programmer
BBS (814) 234-1881

LAI Logical Alternatives Incorporated
Calder Square, P.O. Box 10674
State College, PA 16805-0674
(814) 234-8088

Number **246** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Better C++ Code Generation

by

John M. Dlugosz

Did you ever look at the code generated by a C++ compiler? C++ compilers that generate C code for output are at the mercy of the C compiler they translate for. But a native code compiler generates native code directly from C++ and is not limited to explaining C++ features in terms of C features. There are ways to improve the ideas behind the compiler to produce better code.

Consider parameter passing. In C, all parameters are passed by pushing them on the stack. For large values, you might decrement SP and then do a block copy, but it amounts to the same thing. In C++, values are passed by initialization; that is, the space on the stack that will correspond to the local variable in the called function is given a value in the exact same way an auto variable is initialized. For many types, this requires calling a constructor.

A translator would do something like this:

```
extern foo (mytype x);
// a C++ call
foo (avalue);

//translates to:
struct mytype temp;    /* temporary variable */
mytype_copy_constructor (&temp, &avalue);
foo (&temp);
mytype_destructor (&temp);
```

The translator initializes a temporary variable and passes that. There is no other way a translator could do this. What is shocking is that the native code compiler generates worse code! It makes a copy and pushes the copy by value. The translator passed the copy's address, which was faster. What a native code compiler could do is just construct the thing in the proper place. Consider the following fragment, produced by Zortech 1.07 for MS-DOS:

```
;foo (avalue);
      lea AX,[BP-5Ch]
      push AX          ;address of avalue
      lea CX,[BP-2Eh]
      push CX          ;address of temp
      call __mytype__ctorFRCmytype___
      add SP,4         ;copy made
      lea BX,[BP-2]
      mov CX,017h      ;size of object to push
L28:  push [BX]        ;loop to push the copy on stack
      sub BX,2
      loop L28
      call _foo        ;call foo
      add SP,2Eh
      xor AX,AX
      push AX
      lea CX,[BP-2Eh]
      push CX
      call __mytype__dtor  ;get rid of temp
      add SP,4
```

You can see that the loop to push the copy costs both time and stack space. The translator's output, once compiled, looks the same except that the copy's address is passed instead of its value, eliminating the loop. It would be better if the stack location were the destination of the copy in the first place. This would speed it up, make it shorter, save a large auto variable, and result in a stack frame looking exactly as if a copy were pushed by value, the way Zortech did it:

```
;foo (avalue);
      sub SP,2Eh       ;room for parameter on stack
      mov CX,SP        ;address of this area
      lea AX,[BP-5Ch]
      push AX          ;address of avalue
      push CX          ;address of foo's local copy
      call __mytype__ctorFRCmytype___
      add SP,4         ;copy made
      call _foo        ;call foo
      mov CX,SP
      xor AX,AX
      push AX
      push CX
      call __mytype__dtor  ;get rid of temp
      add SP,4 + 2Eh   ;clear stack from call and the
                       ; variable
```

If the compiler used this sequence instead, the program would run faster, and nothing else about the compiler would need to change. These examples are in the small-memory model. In the large model, the savings are even greater, but the listings are more cryptic. You should get the general idea from the small model.

Returning Values

The inefficiency of return values can be bad, too. In C, the ability to return a structure is relatively new. Upon finding out that it was possible in Microsoft's compiler, I looked at the code produced and decided not to use this feature. In C++ however, it is a common thing to do.

The traditional way of returning a large value is by way of an internal static buffer. The function actually returns a pointer to this buffer, and the caller copies it out. Here is another example from Zortech:

```
;    end of function foo()
;return x;
      lea SI,[BP-2Eh]  ;address of local variable x
      mov AX,DS        ; in DS:SI
      mov ES,AX
      mov DI,offset DGROUP:U0     ;address of static
      cld                         ; buffer in ES:DI
      mov CX,017h
      rep movsw                   ;copy it
      mov AX,offset DGROUP:U0     ;return pointer
                                  ; to buffer
      pop DI                      ;return sequence
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        pop SI
        mov SP,BP
        pop BP
        ret

;     use of function
;avalue= foo ();
        call _foo
        mov SI,AX
        mov AX,SS
        mov ES,AX
        lea DI,[BP-2Eh]
        cld
        mov CX,17h
        rep movsw
```

The function returns a structure. As part of the return process, the value is copied into a static memory area. The address of this buffer is returned, and the caller copies it again to the destination. If the function had not been in an assignment statement but had been part of a larger expression, it would have been copied to a temporary variable (which would then be passed to a destructor after the value was no longer needed).

You can see that this technique is very wasteful: Not only does the multiple copying slow down the program, but the static buffer is used by only this one function, and buffers for many functions can put a dent in the memory requirements of the program.

According to the Zortech manual, the functions that return in this manner are not reentrant, because the static buffer gets overwritten on each call. The manual is incorrect: Because the value is not copied into the buffer until the return statement of the function, and the caller immediately copies it out again, no function can be called while this buffer contains anything. So, contrary to the instructions, such functions are fully reentrant.

A close look will show that one buffer will do for all functions in the entire program. The buffer needs to be large enough for the largest structure returned by value. A great deal of memory could be saved if the compiler adopted a single buffer system. Furthermore, if only one buffer is used, there is no need to have the function return its address in a register: The caller will know where to find it and can load SI with a literal.

Using this buffer technique, you can reduce the amount of buffer memory needed and shave off a couple instructions. But the technique is still rather bad. The obvious improvement would be to change the calling mechanism. The caller could tell the function where it wants the answer by passing an extra parameter that is a pointer to the answer area. This would remove the second copying. Furthermore, the optimization process could ensure that the copy were built there in the first place, eliminating the first copying: If an auto variable is returned (this includes temporary variables), replace its use throughout the function with the use of the return area, and delete its destructor. This would improve speed considerably. The optimizer cannot do that now because of the reentrance problems: The static buffer must be filled just before the return in order to produce a reentrant function.

When Zortech C++ was first released, I pointed out to Walter Bright the cost of functions that return structures. He said he considered passing a hidden parameter to the answer area but decided against it because he wanted to keep the standard C calling convention. This is very costly in performance, because returning structures is so common in C++. Fortunately, future versions of the compiler are supposed to use the better method as well as provide C compatibility when requested.

Bjarne Stroustrup recognized this issue when writing Cfront, and he came up with an interesting compromise: A function that returns a structure is turned into a function that takes a hidden parameter if the translator knows that the function will not be called from C. It knows this will be the case if the value being returned (or any of the parameters) has a copy constructor. A similar improvement is made on all objects passed by value: Instead of copying the object into a temporary variable and pushing the temporary, the definition of the function is modified to pass a pointer to the temporary. This eliminates the loop used to push a large parameter and results in faster code with reduced stack usage.

Because of the differences between Bright's and Stroustrup's policies, a Cfront-based translator will generate better code than Zortech's native code compiler for function calls passing or returning class objects. A native code compiler could use the same techniques found in Cfront, which are architecture independent. However I would expect a native code compiler to do better then this and deal with the stack more directly, as described above.

Calling Conventions

Passing and returning class objects is an important part of C++, and generating good code for these operations does not require much change from methods used by C compilers. Any further improvements, however, do require that the function-calling convention be changed. Why is this an issue? After all, member functions already pass a hidden parameter, and "real" C functions cannot return structures, anyway. Adopting the convention that functions returning structures accept a hidden parameter for a return area would not bother anyone.

But why do we want compatibility with the traditional C calling convention? If I were linking with functions written in assembly language, I would care more about knowing what it is and not quite so much about exactly what it is. In a real program, I want good performance. Making sure the assembly language functions are executed correctly is a very minor part of a program.

Most C compilers now accept special forms of the **extern** statement to allow linkage with functions using different calling and naming conventions. C++ will require such a feature soon, when the new compilers start using type-safe linkage. Why not take this opportunity to break with the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

standard calling conventions along with the naming conventions? If you want to use your "old" assembly language functions, just specify the proper keywords when you **extern** them in the program.

How would you devise a calling convention for C++? Let's start by looking at two common conventions. The C convention is to push parameters from right to left, with the caller popping them off again. The other common technique (commonly called a Pascal call but actually a subset of the Pascal calling requirements) is to push parameters from left to right, with the called function clearing the stack when it returns.

The C convention was devised because in K&R C, no checking was done with parameter passing. Since the function did not know how many parameters were actually passed, it did not clear them; rather, the caller popped off whatever it pushed when it made the call. This was safer all around. Pushing right to left made a variable number of parameters useful; the first parameters are in the same place on the stack, even if the later parameters are missing.

The Pascal technique is more efficient. The program does not need extra code after each function call, and a special form of function return is usually available to clear the stack as it returns. Except for functions that take a variable parameter count, the called function ought to clear the stack.

In C++, "clearing the stack" does not mean just popping the parameters off during the return—it needs to call the destructors for the arguments first. This reduces the size of the program considerably. Rather than destructors being called after every function call, they are in there only once, at the end of the function itself. In this respect, the savings for using a Pascal-style function call is much greater for C++ then for other languages.

Whether the parameters are passed left to right or right to left doesn't mean beans. I suggest they be passed right to left so that they will be consistent with the variable parameter count functions.

C++ has another special consideration for function calls. A member function is not quite like any other procedural language feature. The function needs to know what instance it is operating on. This is usually handled by passing a pointer as a hidden parameter. Looking at the generated code, you see that this makes C++ rather inefficient.

Scope

Languages like C have two forms of scope: global and local. Two addressing forms in the CPU handle this easily. Pascal has a broader meaning of "scope" and sets up pointers on the stack to deal with variables of intermediate scope. This gives Pascal calls more overhead, and accessing these variables of intermediate scope is not efficient, because the pointer must be loaded first (as a variable of local scope) and the proper variable then must be found offset from it.

C++ has a novel idea of scope, but you can see it is not much different from Pascal's problems. The way it is usually accomplished is by passing a pointer as a hidden parameter. To access a member variable, the CPU first accesses the pointer (known as **this**) as a local variable and finds an address offset from it. While references to intermediate variables in Pascal can be avoided, references to member variables in C++ are extremely common.

I suggest that the **this** pointer be kept in a register. The instance would be passed in the register when a member function was called. It is common to have a member function call other members. In this case, the register is already loaded, and nothing further need be done. If a member of a different variable were called, the register would be saved by the caller before loading it with the new value. Some MS-DOS C compilers, such as WATCOM, can generate code to pass the first parameter in a register. Using such a compiler with a C++ translator can yield fantastic results.

C++ really has three scopes: global, local, and member. In a 68000-family CPU, one of the address registers can be reserved for the member pointer. All member variable references are relative to it. It is still set correctly when another member function is called; it is pushed and popped only when a different instance is being worked with.

In the 80X86 family, things are more difficult: There simply are not enough registers. The globals are absolute offsets in the data segment, and the locals are relative to BP in the stack segment. There is no corresponding segment for the member scope. I would suggest using DI in the small model. It is commonly used for register variables, and what makes a better register variable than the **this** pointer? For the large model, things are more complicated. ES:DI could be used, but ES is needed every time any pointer is used. SI:DI could be used and SI moved to ES when needed. A more radical solution might consider using DS, since global-variable accesses are rare and member-variable accesses are common.

A similar procedure can be used for functions that return a structure. The address of the destination should be supplied by the caller, and the function will place it there. This can be done on the stack, but a register might be better. In the 68000 family, one of the address registers can be reserved for this purpose. In the 80X86 family, SI could be used. (Even

The C Programmer's Companion
Now Available for OS/2 !!!

***CCSYM** - C Symbol Compiler*

* Analyzes ANSI C source code. Microsoft/Turbo extensions supported.
* Automatically generates separate commented function prototype files for all global and local (static) functions.
* Log File lists all your symbol definitions with detailed usage statistics.
* Generates a "symbol database" which CCRIDER uses to quickly retrieve ANY symbol definition in your application, no matter how large it is.
* Incremental compilation supports efficient database updates.

***CCRIDER** - Pop-up Symbol Lookup*

* Pop-up utility which converts your own editor into a powerful hypertext programming environment.
* Point at a symbol in your editor, then pop-up its definition in a window.
* One keystroke "Edit" command allows you to edit the definition in your own editor, no matter which file or directory it was defined in.
* "Paste" command inserts the definition into your currently edited file.
* Supports multiple editing sessions under OS/2.

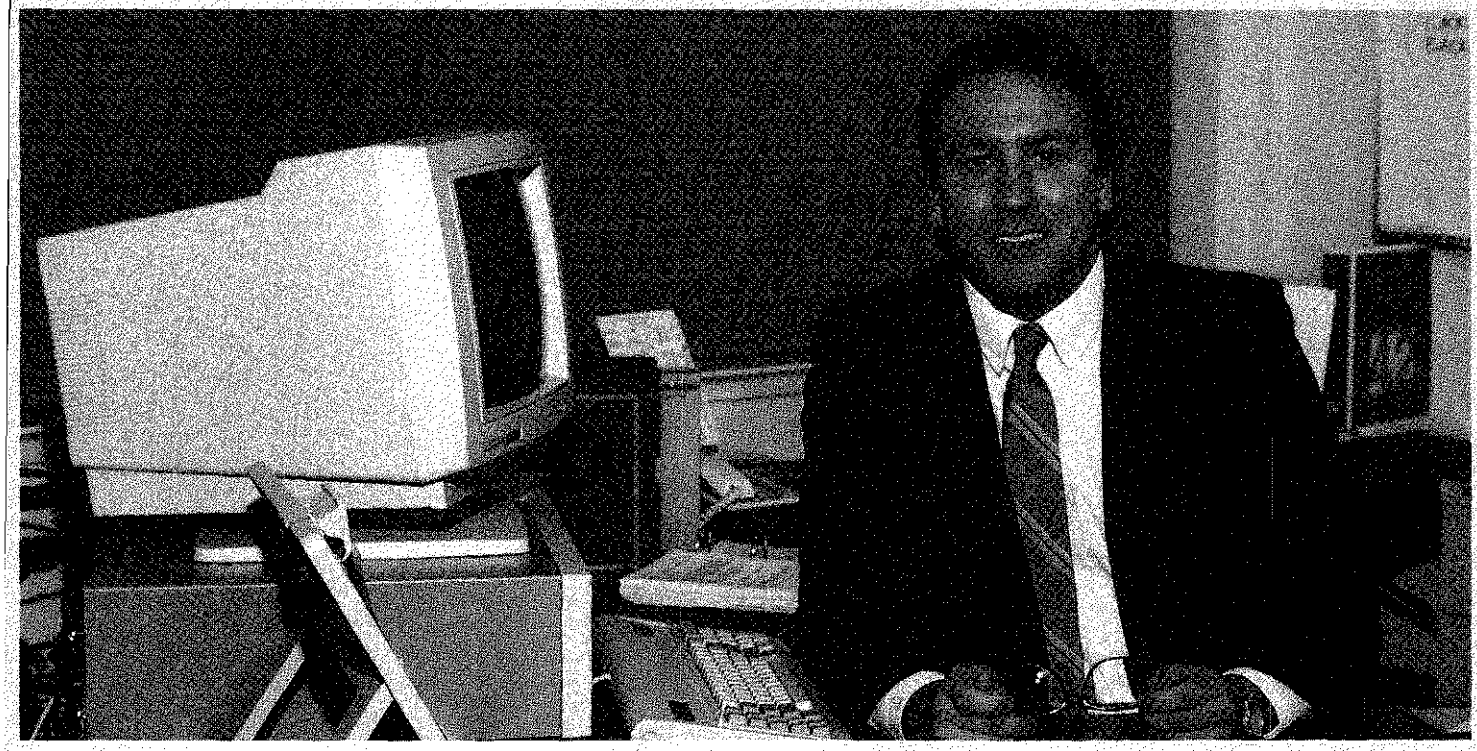
DOS Version $89
OS/2 Version $179

FREE DEMO

Western Wares
Box C Norwood, CO 81423
(303) 327-4898

Number 100 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



When it comes to selecting compilers, MetaWare is the right choice!

Developing the next generation of software products is serious business. You need the best tools to produce the best code. You need **MetaWare**.

Superior Compilers

A compiler that handles large programs with ease, while producing the expected results, is the key to developing the highest quality applications. Many of **MetaWare's** customers say that **High C** ™ and **Professional Pascal** ™ are the highest quality compilers in the industry. They are reliable, and well documented, and their superior diagnostic messages help to produce better products more quickly. **There are no surprises with MetaWare compilers.**

Compiler Features

- ANSI Standard with extensions
- Generates small, fast executables
- Support of 80x87, Weitek 1167/3167, 68881, and Am29027 math co-processors
- Global common subexpression elimination
- Live / dead analysis
- Constant propagation, copy propagation
- Tail merging (cross jumping)

Multiple Platform Support

To support multiple platforms effectively you need the same compiler on each, eliminating the time and cost of developing and maintaining a new version of the product for each platform. **MetaWare** is *the only* company that provides the same compilers on a wide variety of platforms and operating systems, to both OEMs and End Users.

MetaWare uses its own **Translator Writing System** to create all of its compiler products. Common components and library functions are shared across the product line. Improvements to compilers are immediately realized on all platforms.

Supported Platforms

- 386: Extended-DOS/UNIX/XENIX
- PC: DOS/OS2
- Sun-3, Sun 386*i* , Sun-4,
- IBM: AIX/386, AIX/RT, AIX/370
- DEC VAX: VMS (host for cross compilers and TWS, not native)
- DEC VAX: Ultrix
- AMD 29K (Native/Cross)
- Intel i860 (Native/Cross)

MetaWare's Commitment

MetaWare's unique approach to business is its ability *and willingness* to support both OEM and End Users alike. The commitment to providing quality products doesn't end when you receive your compiler. Technical phone support is provided by engineers at a level of expertise consistent with the quality of **MetaWare's** products.

So when it comes to selecting a compiler company, **you need the best!**

Can you really afford anything less?

MetaWare™
INCORPORATED

Compiler Products for
Professional Software Developers

2161 Delaware Avenue
Santa Cruz, CA 95060-5706
Phone: (408) 429-META (6382)
FAX: (408) 429-WARE (9273)

MetaWare, High C and Professional Pascal are trademarks of MetaWare Incorporated; UNIX is a trademark of AT&T; Sun-3, Sun-4, and 386*i* are trademarks of Sun Microsystems, Inc.; PC-DOS, RT PC, and IBM are trademarks of International Business Machines Corporation; 386 and i860 are trademarks of Intel Corporation; DEC, VAX, Ultrix, and VMS are trademarks of Digital Equipment Corporation; XENIX is a trademark of Microsoft Corporation; 1167 and 3167 are trademarks of Weitek Corporation; 29K and 29027 are trademarks of Advanced Micro Devices, Inc; 68881 is a trademark of Motorola Corporation.

PROTECTED BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

in the large model, the stack segment can be assumed.) In large-model member functions, it would need to use the stack.

Let's take a look at how these improvements would affect the examples given earlier:

```
Call with copy constructor

;foo (avalue);
     sub SP,2Eh        ;room for parameter on stack
     mov DI,SP         ;address of this area
     lea AX,[BP-5Ch]
     push AX           ;address of avalue
     call __mytype__ctorFRCmytype___
     call _foo         ;call foo

Function returning a structure
;    end of function foo()
;return x;
; there is nothing to do, since the optimizer
; replaced all uses of local variable x with
; [SI]. The answer is already there.
      pop DI           ;return sequence
      mov SP,BP
      pop BP
      ret

;    use of function
;avalue= foo ();
      lea SI,[BP-2Eh]    ;where I want the answer
      call _foo
```

You can see even from these simple examples that the program compiles *much* better.

Another area where a native code compiler can't compete with a translator is static constructors and destructors. A native code compiler can generate code for static constructors (and destructors) that does not require the object or executable code to be patched. It can also handle constructors for static variables inside functions much more easily.

Consider three code segments, **a**, **b**, and **c**, defined in that order in a module in the library, with **a** containing a label **static_ctors**, **b** empty, and **c** containing a **RET** statement. Each translation unit generates its static constructor code in segment **b**. When the program is linked, all the static constructor code is sandwiched together between the label and the return statement. The startup code calls **static_ctors**. An analogous method can be used for the destructors.

Caveat Vendor

Native code compilers for C++ need to deal with more issues than C compilers. They can generate better code if they understand the C++ constructs, rather then expressing them in terms of C constructs internally. C++ can produce much better code than is seen in current implementations. Since C++ expresses higher-level ideas, the compiler can have a better idea of what is intended and generate better code then the equivalent C program.

I am afraid that the round of C++ compilers soon to appear from vendors of popular C compilers will not understand this and will generate lousy code. Vendors, listen to me! If you want the first round of published benchmarks to show code that is a third the size of and twice as fast as the competition's, look at the above examples carefully.

John M. Dlugosz is a programmer, consultant, and writer based in Plano, Texas. He is working on his first book, Advanced C++ Programming.

PROTECT INTELLIGENCE WITH INTELLIGENCE

PROTECH KEYs:

- ❑ are totally transparent to the end user
- ❑ allow unlimited back up copies
- ❑ free up disk drives
- ❑ do not interfere with peripheral operations
- ❑ are easy to install
- ❑ protect software developers' revenues
- ❑ provide the highest level of protection available

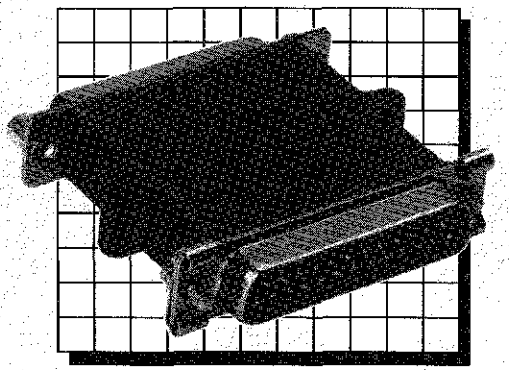
PROTECH's KEYs provide the highest level of protection because of:

- ❑ assembler based routines (not drivers)
- ❑ encrypted routines
- ❑ physically unique hardware KEYs
- ❑ sophisticated software installation advice

THE KEY

The KEY provides effective software protection while insuring customer satisfaction. The KEY is a random response device which is designed for identically reproduced software packages.

SOFTWARE PROTECTION KEYs BY PROTECH



WE HAVE THE *KEY* TO SOFTWARE COPY PROTECTION

For a demonstration package or additional information, please write or call:

1-800-843-0413

THE MEMORY KEY

The MEMORY KEY is a programmable software protection device. Each byte of memory can be addressed in groups or individually. Possible applications for the MEMORY KEY include:

- ❑ modular package control
- ❑ serialization
- ❑ customization
- ❑ access control to PCs

Information stored in the MEMORY KEY can also be altered by your application software in the field. This dynamic writing to the MEMORY KEY opens up unlimited possibilities. Possible applications for the dynamic read/write function include:

- ❑ demo control
- ❑ software leasing
- ❑ updating modules in the field
- ❑ any "counter" operation

PROTECH MARKETING, INC.
1804 W. Southern Parkway • Building A-112
Durham, North Carolina 27707
(919) 490-4970 FAX (919) 490-4974

Number **228** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FOR DEDICATED DEVELOPERS WHO PREFER TO DEDICATE LESS TIME TO CODING.

LEAVE THE CODING TO US.

PRO-C will give you more time than ever before to spend on system design.

PRO-C...the professional C code generator you need for screen programs, menu programs, report programs and multi-file updates.

There are no run times and no royalties. PRO-C interfaces with most popular DBMS's and is even shipped with source code to our libraries.

It's also the only source code generator that has versions available for MS-DOS, UNIX, XENIX and QNX*.

PRO-C. It's dedicated to helping you make the best use of your valuable time.

Order today. Call our toll-free number: 1-800-265-2682.

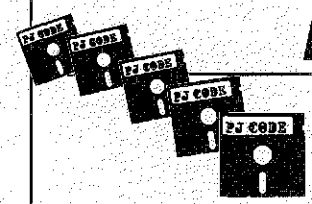
VESTRONIX

Allen Square, 180 King St. South, Suite 550, Waterloo, Ont. N2J 1P8 • 1-800-265-2682

**PRO-C is a registered trademark of Vestronix Inc. MS-DOS, UNIX, XENIX and QNX are registered trademarks of their respective companies.*

Number **287** on the Reader Service Card

PJ Code Is Available!



On disk from:

Programmer's Journal
P. O. Box 30160
Eugene, OR 97403
800-234-0386
503-747-0800 in Oregon
All Code Disks $10.00 each.
To order, see ad on page 88.
PJ Code DiskScriptions
are available for only $39.95.
To order, see ad on page 8.

On BBS from:

FIDO/OPUS Network
Blazer Net 152 Node 9
503-343-9724

Generation Five BBS
(Voice) 301-495-6690
(Data) 301-495-2932
Accessible via PC Pursuit, Washington D.C. From Files submenu, select "P" for Personal Downloads.

Gilmore Systems of Southern California
5447 Indian Hills Drive
Simi Valley, CA 93063
(Voice) 805-582-1360
(Data) 805-582-9306

Ithaca Street BBS
1145 Ithaca
Boulder, CO 80303
(Voice) 303-494-8865
(Data) 303-494-8868

PCS-49 on CompuServe
Section 9
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
(Voice) 800-MISOSYS

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention *PJ* when you call.

Icon-Tools

- 2, 4, 8, or 16 colors per icon.
- Up to 32000 icons per file.
- Icons as large as the screen.
- Editor with dozens of functions, read PCX, MSP, IMG files, colorize B&W images, resize icons, and much more.
- Function library supports icon drawing, digital dissolve, scrolling, and animation.
- Full mouse support with 30 different cursors, functions for icon menus.
- Hundreds of 16-color icons included.
- Many uses: clip art, fonts, publishing.
- Reasonably priced at $150; NO royalties.

Menuet

- A true professional-quality graphical interface for DOS applications.
- Movable and resizable windows, pull-down menus, dialogue boxes, text entry forms, radio buttons, scrollable lists, signal boxes and guages, scrollable displays, rubber-banding functions, more!
- Automatic window management; screen save/restore uses conventional memory, EMS (expanded) memory, or disk paging.
- Interface Design Tool for interactive editing and prototyping of applications.
- $250, NO royalties, source available.

Baby Driver

- Print graphics on 200 different dot matrix, inkjet, and laser printers.
- Supports color printers or render color as shading patterns on B&W printers.
- Scale, rotate, and position images.
- Includes PCL support for downloading HP soft fonts, positioning and sizing text, and simple printer graphics.
- Print from off-screen in-memory images.
- Compact function library links with your application-- no TSR's.
- Device Table editor and Install utility.
- Only $250 and NO royalties.

Font-Tools

- Font support for DOS graphics programs, use with any product.
- Font editor for creating, editing fonts-- make italic, bold, 3-D, etc.
- Utilities for modifying, viewing fonts, download to printer, convert soft fonts.
- Complete function library - control text justification, drawing direction, fg and bg colors, dynamic underline and bold.
- Draw text into program-supplied buffers.
- Over 75 sample font and icon files in a variety of styles and sizes.
- $150 and NO royalties. Source for $100.

ISSI Ithaca Street Software, Inc.
1145 Ithaca, Boulder, CO 80303

Quality that's Genuine

Telephone 303-494-8865
Fax 303-494-7802 BBS 303-494-8868

Number **226** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A Library System for Assembly

by

John Otken

Libraries are essential tools for efficient assembly language programming, yet they are rarely used. Over the last several years, I have developed an automated system that simplifies the use and maintenance of libraries.

Since assembly language libraries are uncommon, we might review functions of libraries under the C language. A primary purpose of C libraries is to provide a rich selection of general-purpose routines that permit programmers to work more quickly through the use of pre-existing code. These routines, which are listed in the reference section of your manual, include functions for input, output, string operations, memory allocation, and various conversions. Turbo C documents more than 400 such routines.

A second purpose is to provide "hidden" routines for compiler support. For example, Turbo C does not know how to multiply two long integers, but it does know which library routine to call—in this case, LXMUL@. These compiler-support routines are undocumented and never called directly by a program.

```
long a=1, b=2, c;
c = a*b;  /* generates call to LXMUL@ */
```

Although an assembly language library has no requirement for undocumented routines, it should provide parallel functions of both the general-purpose and support categories. An advantage of moving support routines to a library is the reduction of clutter in the various modules; program code becomes focused.

A final purpose of any library is to provide a convenient way to collect multiple object files into a single file. At link time, only referenced modules are included in the program image. Not only does this use storage more efficiently, but it also helps eliminate routines from programs that are not referenced.

Most programmers consider a C compiler useless without its library. An assembler is no different, and the lack of a library is why many programmers use assembly language only to write subroutines for their C code. Providing a library would open many new applications for assembly language. The first step is to set a realistic goal. Whereas C defines up to five separate memory models and a library for each, I define a single assembly language model, a hybrid design combining the "small" C model with a large, dynamic storage space. Obviously, one library is much easier to program than five.

The next step is the use of a library maintenance system that automatically reassembles the library file and creates documentation on member routines. If you have ever tried to add routines to your C library, you will greatly appreciate this system; you might even try adapting the programs for use with C.

Unfortunately, I am not offering a rich selection of routines for your library, and, to my knowledge, no worthwhile library is available for purchase off the shelf. My personal library, however close to nirvana, is not available, although over time I will make portions available via this column. Check *PJ* 7.4 (July/August 1989) for some simple **outchr** routines and C-like **isdigit** and **strlen** functions.

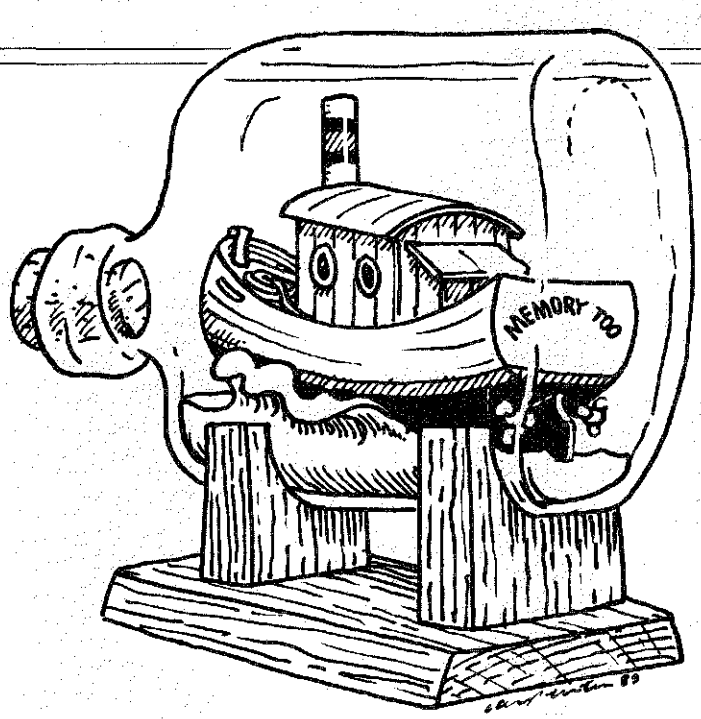
In early development of this system several years ago, I wanted the capability to enhance and expand the library during the normal course of programming. The ability to "cut" a general-purpose routine from a module and "paste" it into a source template would permit the selection of routines to grow with minimal effort.

The first component of this system is LIBGEN, which writes the command files required to assemble updated source files and generate the actual library file. The program was initially designed to work with Microsoft's assembler and librarian, although I have enhanced it to work with the superior products offered by SLR Systems: OPTASM and OPTLIB. I have not tried this program with Borland products, but I presume that you can also use TASM and TLIB.

LIBGEN is a short C program that builds a list of the assembly language source files in a directory, sorts the list, and writes a MAKE file for MASM, a pseudoMAKE file for OPTASM, and a response file for LIB.

If you use Microsoft or Borland products, a batch file invokes LIBGEN and then runs MAKE, which invokes the assembler for those modules that require reassembly. The batch file deletes the current library file, LIB.LIB, and executes LIB to completely rebuild the library. Here is the MAKLIB batch file:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED




```
libgen *.asm
make lib.mak
if errorlevel 1 goto done
del lib.lib
lib @lib.arf
:done
```

The two annoying aspects of this method are that the OBJ files must remain in the directory at all times and that invoking the assembler for each module is slow. With more than 150 modules, my directory does not need the clutter of excess OBJ files, nor do I have the patience to watch an assembler execute 150 times when the master include file changes.

Fortunately, OPTASM and OPTLIB offer excellent solutions to both problems. OPTASM has a built in MAKE-like ability; not only do you not have to reload the assembler, but it even automatically buffers include files so that the assembly process proceeds at an incredible rate. OPTLIB has a useful, time-based replace feature that compares the creation time of an object file with the date stamp of the object file within the library and updates modules as required. Here is the MAKLIB batch file for use with OPTASM and OPTLIB:

```
libgen *.asm
optasm @lib.opt
if errorlevel 1 goto done
optlib lib -*;
del *.obj
:done
```

In one minute, OPTASM and OPTLIB completely rebuilt my library of 157 modules containing more than 14,000 source lines.

The source for the LIBGEN.C program appears in Listing 1. You can compile the program with either Microsoft or Turbo C. The program assumes source files are dependent on a LIB.EQU file, which is not covered in this column; for now, create a dummy LIB.EQU file. You must create a separate directory for your library source files, because LIBGEN assumes that all files in the current directory with an extension of ASM are part of the library.

The second component of this library system is a program that automatically generates documentation for the routines in the library. When a library has 600 entry points in 150 modules, you waste too much time searching for function definitions. A more efficient approach is to collect and organize the essential documentation for each routine into a single file.

The ASMHDR program performs precisely that task. It reads a list of assembly language source files, extracts the header comments, sorts the comments alphabetically, and writes the comments to a file. The resulting file is easy to reference while programming. I have a key defined in my editor to load the comment file; I just press the key and use the search feature of my editor to find the header comment for a particular function.

Recall that the previous column introduced header comments, which contain the important information about a function—namely the register entry and exit conditions. In the following example, the function **strskp** accepts a pointer to a string in registers DS:SI, returns a pointer to the byte past the NULL delimiter in SI, and bashes whatever was in the AX register:

```
;;    strskp
;
;     entry    DS:SI   string address
;     exit     SI      updated past \0
;     uses     AX
;
strskp proc
     lodsb
     cmp    al,0
     jne    strskp
     ret
strskp endp
```

The header comment provides a good description of the operation of this function and presents the information in an attractive and easy-to-read format.

The ASMHDR program is very simple, containing no logic to process the complex assembly language syntax defined by MASM. The program scans a text file for lines that start with a double semicolon followed by a tab. From that point, it collects text for all subsequent lines until a line begins with a character other than a semicolon. After collecting all such blocks of text from the entire list of files, ASMHDR sorts the header comments and writes them in order.

ASMHDR's limitation is that all header comments are processed, even those for routines that are strictly internal to a library module. An intelligent solution would involve reading PUBLIC statements and processing only those headers that appear in a PUBLIC, but that is not yet the case. In the meantime, you can easily defeat ASMHDR's search

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

algorithm by altering the header comments that are not to appear in the document file:

```
;;*   internal lib routine
;
;     entry  AX  ...
;
```

Placing a "splat" character after the double semicolon will cause ASMHDR to skip this routine. One reason I reserve the double-semicolon syntax for header comments is to simplify programs such as ASMHDR. There are many such useful utilities you could create to make programming easier, and unless you want to write your own assembler, I suggest keeping the syntax of programs as simple as possible.

The source for the ASMHDR.C program appears in Listing 2. You can compile the program as is with Turbo C. Microsoft C users must change the name of an include file and comment out the **printf** statement that references the **coreleft** function.

The third component of the library system is a template file. To create a new module, substitute a title and procedure name for each x, fill in the header comments, drop in code for the procedure, and save the file with an appropriate name.

```
        title   x
        .model small
        .code
        public  x

;;      x
;
;       entry
;       exit
;       uses
;
x proc
x endp
        end
```

The template file is simple, but it saves time and keeps you from having to break your train of thought to create a new module.

This template does not reference a master include file, nor does it have a place for static data definitions. If possible, library routines should avoid static storage, because it limits the usefulness of the library for applications like multitasking. Ideal library routines are independent of memory models, with the exception of a near or far return instruction.

Using the Library

BEEP is as simple as programs get: With a press of Ctrl-G, it beeps the speaker and returns to DOS, calling two library routines, **outchr** and **ms_dos**, in the process. OUTCHR.ASM and MS_DOS.ASM appear in listings 3 and 4. Enter the two library routines, and execute one of the MAKLIB batch files from above to build the LIB.LIB library file. In a separate directory, enter BEEP.ASM:

```
        title   beep
        dosseg
        .model  small
        .stack
        .code
        extrn   outchr:near,ms_dos:near

;;      main
;
main    proc
        mov     al,7
        call    outchr
        mov     ax,4C00h
        jmp     ms_dos
main    endp
        end     main
```

I call or jump to an **ms_dos** routine instead of using the shorter, two-byte INT instruction because the technique is a useful debugging aid and a front-end routine can replace **ms_dos**, if necessary.

After entering the source for BEEP.ASM, assemble and link the program with commands similar to the following (substituting the names of your assembler and linker for "asm" and "lnk", respectively):

```
asm beep;
lnk beep,,,\lib\lib.lib;
```

You now have a BEEP program. Switch to the library directory, and run the ASMHDR program with

```
asmhdr lib.doc *.asm
```

Examine the LIB.DOC file; it contains the header comments from MS_DOS and OUTCHR.

This completes the automated library-maintenance system. I will add routines to this library on a regular basis, but you are responsible for the majority of library growth.

Register Usage

The increased performance of assembly language programs is due in part to important values remaining in the registers more often than they would in code generated by language compilers. This permits the use of register-to-register instructions, which are at least twice as fast as their register-to-memory equivalents.

Contrary to the claims of computer scientists and authors of high-level languages, MS-DOS compilers are not going to produce code that approaches the performance of even decent assembly language. Code generation is too difficult for the 80X86 architecture. When a programmer constructs an algorithm in a C program, he codes for an abstract machine that has no real-life equivalent. The compiler's job is to convert an algorithm from the abstract machine defined by C to the real hardware defined by the 80X86. This is essentially a conversion process, and—like every other conversion process in this world, from a nuke to an air conditioner—it operates at efficiencies much less than 100 percent. So, when you hear someone talk about a compiler that generates code superior to that of assembly language, ask him if the compiler comes with a free perpetual-motion machine.

One difference between good and bad assembly language is the effective use of the rather limited architectural features of the 80X86. To achieve this goal, we select an efficient memory model and define a strategy for the use of registers.

The 80X86 architecture offers seven 16-bit general registers. Intel labels the stack pointer an "eighth" general register; it never is. Each of the seven general registers has one or more dedicated instructions that manipulate or access the register contents:

```
AX      cbw, lods, stos
BX      xlat
CX      loop
DX      mul, div, in, out, cwd
DI      stos
SI      lods
BP      enter, leave
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Because of these register-specific instructions, it is important to choose the correct register to hold a value so that when you manipulate the value, you can take advantage of the special instructions.

A most obvious example is a loop index. If you are going to do something *n* times, *n* is best kept in the CX register. This permits the use of the LOOP instruction, which is faster and more readable than its two-instruction equivalent:

```
        mov   ah,0          ; checksum record
        mov   cx,[si-2]     ;  (assume cx!=0)
chk1:   lodsb
        add   ah,al
        loop  chk1
```

The three kinds of data most commonly held in registers are numbers, characters, and addresses. Numbers can be 8, 16, or 32 bits (as in C). A number's signed or unsigned characteristic is not important for assembly language data except for relative comparisons. Because the 80*X*86 has 16-bit registers, that is the most common numeric data size.

An address pointer on the 80*X*86 consists of two 16-bit portions: a segment and an offset. Depending on the circumstances, it may be more convenient to think of an address as a full segment–offset pair or just as an offset. The full segment–offset specification is useful for pointers to dynamic storage. A lone offset suffices when the segment is known (such as DGROUP) or when indexing an offset with another pointer.

Indirect addressing is possible only from the four index registers (BP, BX, DI, SI), which dictates that these registers tend to carry addresses while the three remaining registers (AX, CX, DX) tend to carry numbers. In spite of the official Intel nomenclature, I often refer to the base registers BX and BP as *index registers;* don't be confused.

The closest thing I can offer as a systematic method for register assignment is to ask yourself what kind of quantity you are working with and try to find an approximate description below. Through experience, you gain an instinctive feel for the register set and know the optimal way to express algorithms.

Below are guidelines for register usage that work very well in my programs. By following these conventions, you may well find your programs shorter and more efficient. Remember, if the preferred register is not available and another register is free, use the free register.

- AX is for computation of all kinds of temporary values, including the least-significant half of a numeric dword. Character values rarely appear in registers other than AL or AH.
- AX is a favorite register for passing and returning numeric quantities from a subroutine; however, note the preferred use of CX and DX below.
- AX is not suitable for long-term storage across one or more subroutine calls, because subroutines often return a value in AX or neglect to preserve it.
- BX is the least useful of the index registers because, unlike the valuable features of DI, SI, and BP, BX's only claim to fame is an otherwise useless XLAT instruction. In a later column, I will introduce sophisticated techniques that use BX; for now, consider BX a general-purpose register.
- The LOOP instructions designate CX as the counting register, and I keep all kinds of counts in CX, even those inappropriate for LOOPing; as examples, a

SOFTWARE DEVELOPERS: ARE YOUR PRODUCTS TOO HARD TO INSTALL?

In today's competitive market, your product **must** be easy to use -- immediately. It is no longer commercially acceptable to tell the end-user to use batch files, COPY, DISKCOPY, RESTORE, or to require wading through paragraphs of text in the manual. The installation process is something that a program developer **should** and now **can easily** completely automate.

INSTALL 2.3 is a bullet-proof automatic installation program you can distribute - royalty free - with your product • Features: Up to 4.3 gigabyte file sizes • Windowing/color user interface • **Full C source** with project, make, link, and batch files • File integrity verification • Intelligent handling of virtually all errors (open disk drive doors, full disks, unformatted disks, etc) • Hard & floppy disk installation • Multiple source & target disks • Multiple target subdirectories (which *INSTALL* will create if necessary) • Unlimited free tech support • Granular installation • Easy method to add-in new functionality • Ability to execute DOS commands • Can intelligently modify CONFIG.SYS & AUTOEXEC.BAT files • Includes **data compressing file librarian** • **NEW** Automatically detects and can base the installation on the CPU (86/286/386), DOS version, display (CGA/B&W/MONO/EGA/VGA), NetBIOS, ANSI.SYS, LIM, etc.

INSTALL offers 3 years of proven reliability that has been used to install over 500,000 copies of products including some of the most prestigious in the industry. We would like to add your name to that list.

ORDER NOW TOLL FREE 1-800-331-2783

KNOWLEDGE DYNAMICS CORPORATION
Highway Contract 4 Box 185-H, Canyon Lake, TX 78133-3508
Voice: 1-512-964-3994 FAX: 1-512-964-3958
MasterCard/VISA/COD/Purchase Order **$149.95**

Number **295** on the Reader Service Card

ASMFLOW **$99.95**

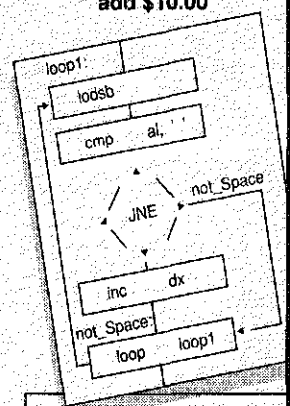
S/H $3.00
Outside U.S.
add $10.00

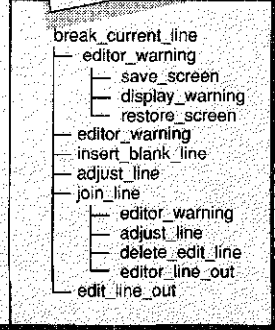
AT LAST!

YOU CAN STEP BACK AND TAKE A LOOK AT YOUR CODE

FLOW CHART AND ANALYZE YOUR ASSEMBLY LANGUAGE SOURCE CODE

- Flow Charts
- Tree Diagrams
- Stack Sizing
- Register Analysis
- CPU Timing Analysis
- Procedural X-Reference
- 8088/87 to 80386/387
- Context-Sensitive Help
- Menu/Batch/Command line Operation
- MASM 5.1 Compatible





QUANTASM CORPORATION
19855 Stevens Creek Blvd, Suite 154
Cupertino, CA 95014
(408) 244-6826

VISA • MC 30 - Day Money Back Guarantee

Number **252** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

memory-allocation routine accepts a byte count in CX, and the **strlen** function returns string length in CX.

- On rare occasions when manipulating a pair of 32-bit values, use DX AX as the primary register and BX CX as the secondary. Consistency is important: Always keep the most-significant words in DX and BX and the least-significant words in AX and CX.
- DX is good for general-purpose numeric storage, especially when calling subroutines that might bash AX. DX serves as the most-significant half of 32-bit numbers in DX AX.
- DX might also contain a separate count when CX is busy. For example, DX might count up the number of words in a string as CX counts down the number of characters left.
- Another use for DX is to point to the end of a buffer. While we cannot use DX as an index register, we can compare an index register such as SI or DI with DX. This is valuable when processing binary information, where end of data is not easily determined from the data itself.
- The primary purpose of DI is that of output pointer. The MOVS and STOS instructions that use DI are especially useful for "write-once-maybe-read-later" operations, such as writing to a display buffer or an output file.
- DI is also useful as a general-purpose index register, although not quite so useful as SI. On occasion, ES:DI will point to a static string, especially when DS:SI is not available, as might be the case in a search routine.

The Indispensible Quick Reference Guide for Assembly Language Programming

MASM QuickRef

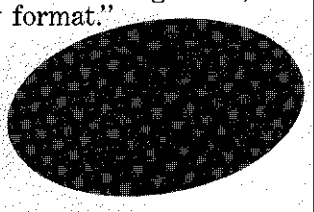
- MASM Instruction Set
- Video Interrupts
- Keyboard Scan Codes
- Display Attributes
- Shift Mask Codes
- Hex-Dec Binary Conversion

FREE Demo Disk Available

Now Also Includes Programmer's Calculator

Memory Resident — Pop-Up Windows

"It's a friendly tool that lets you quickly find commands and procedures in an organized, concise pop-up window format."



Ultra-X, Inc.
2005 De La Cruz Blvd.
Suite 115
Santa Clara, CA 95050
800-722-3789
In CA: 408-988-4721
Fax: 408-988-4849

Number **290** on the Reader Service Card

- When not required for output or as an index register, DI is suitable for computing or holding numeric values. Avoid using DI to pass or return numeric values to procedures.
- It is easy to become confused about which segment register associates with DI. Depending upon the addressing mode, three different segment registers combine with DI to form an address. The MOVS and STOS instructions use ES; DI indexed alone or with BX uses DS; DI double-indexed with BP uses SS.

```
STOSW           ; uses ES
MOV AX,[DI]     ; uses DS
MOV AX,[BX+DI]  ; uses DS
MOV AX,[BP+DI]  ; uses SS
```

- Of the four index registers, SI is king. Whenever reading strings or other data, try to use SI as the pointer. The LODS and MOVS instructions make read access via SI particularly convenient and efficient.
- Because a program is almost always processing some sort of input data, SI is rarely used as a general-purpose register for numeric values.
- A very important characteristic of BP is its ability to address memory via the stack-segment, as opposed to the data-segment, register. High-level languages use this feature to access parameters and automatic variables on the stack. Although this is a useful feature, our primary use of BP is to access static variables and DGROUP.
- While making decisions on register assignments, spend more time designing register usage for the entry and exit conditions of subroutines than on the code sequences within a routine. A primary feature of subroutines is reusability, and the more often a program references a routine, the greater its influence on the program. Poorly designed entry and exit conditions degrade code around each call to the routine and can ripple through several logical layers. A poor selection within an isolated sequence of code affects only that section of code.
- If efficiency dictates use of a less-than-ideal register within the logic of a routine, use the ideal register in the entry/exit conditions and the necessary register within the logic of the routine. One or two move instructions is less of a penalty than poorly designed routine.

Assembler vs. *Assembly*

An article in the June issue of *Dr. Dobb's Journal* reminds me of an important piece of lexical knowledge for programmers: Assemblers assemble assembly language—assemblers do *not* assemble assembler. I'm referring to the misuse of the word *assembler*. A programmer programs "in assembly language," which one might shorten to "in assembly" but never "in assembler." Got it?

The article also inspires me to note an error of my own, one of omission. In my last column (*PJ* 7.4), I warned against the practice of commenting every line of code, because the excess comments often provide no additional information and obscure useful documentation. I neglected to write that use of too many remarks frequently results in comments that are out of date or incorrect. For example, on page 149 of the same issue of *DDJ*, the following code appears:

PRODUCED 2006 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Software Engineering Store

C

Borland Turbo C v2 ..... $105
Borland Turbo C v2 Prof. $175
Blaise C Tools Plus/5.0 ..... $99
Creative Vitamin C $169
Genus PCX Programmer's Toolkit ... $155
read, write, modify & modify ".pcx" graphics files
Gimpel PC Lint $99
Island Systems C Meta-Menu $165
Magna Carta C Windows Toolkit $89
Microsoft C v5.1 ..... $325
Microsoft Quick C $73
Novell Btrieve $199
Softfocus Btree/ISAM $99
comprehensive file/index manager with all source
Wordcraft C Workshop $69
C training manual & software
Zortech C++ $129
Optimizing C++ Compiler

Other Languages & Tools

Abraxas PCYACC $359
Digitalk Smalltalk/286 ..... $164
Borland Assembler & Debugger $105
Borland Turbo Pascal v5.5 Pro $175
Burton Tlib $89
source code management system
Copy Tech Disk Duplicator Pro $129
Matrix Layout $129
Microsoft Macro Assembler $111
Microsoft Windows SDK v2.1 ..... $349
Mortice Kern MKS Toolkit v2.3 ..... $175
KORN shell & 130 UNIX™ -like utilities
Periscope I 512K v4 ..... $599
Polytron Professional PVCS $359
Qualitas 386-Max $69
80386 memory manager
Solution Systems Brief ..... $call
Turbo Power Turbo Professional $99
Whitewater ACTOR $449
High productivity development for MS Windows

Free Catalog & Newsletter

We carry hundreds of products from dozens of publishers. Call or write for your free subscription to our catalog and newsletter.

Returns

We want you to pay for only the tools you really use. If you order anything featured in this ad (or most of the products in our catalog) you can open and use the product for up to 30 days and still return it if it doesn't meet your needs.

386 Hardware

Hauppauge 386XT replaces your XT motherboard to provide top 80386 performance with maximum compatibility. The 386XT works with virtually all 8-bit and 16-bit adapter cards and runs standard versions of 16-bit and 32-bit software including DESQview, OS/2, Windows/386 and UNIX. The board includes 16Mhz 32-bit CPU, 1MB zero wait state memory, 80387 socket and 8 slots (2 16-bit, 1 32-bit)

The 386XT is backed by a 30 day money-back guarantee and one year warranty.

386XT w/1MB RAM $1295
20Mhz AT w/1Mb $1645
33Mhz AT w/4MB $3875

Hauppauge!

The Software Engineering Store

1-800-443-7176

In Calif: 1-800-635-0969

(415)957-0111

Visa • MasterCard
COD • Check

- Most orders shipped within 24 hours
- Free shipping on orders over $100

657 Mission Street, #300-PJ
San Francisco, CA 94105

Programming Classes

The Software Engineering Store offers hands-on training classes in C, Pascal and other languages at our San Francisco store. Please call for a course outline and schedule.

C Windowing

Vitamin C $169
VC Screen $129
top-notch code generator for Vitamin C
DESQview API Ref & C Lib . $179
use internal DESQview functions
Vermont Views $359
complete functionality & great documentation
DataWindows $229
easy-to-use with great documentation & full source
C Windows Toolkit $89
high performance low level functions

Debugging

Periscope v4 *by Periscope*: The Periscope debuggers provide the combinations of hardware and software you need to solve really tough programming problems. Periscope includes all the debugging features you need, but with minimum overhead and maximum flexibility. With Periscope you can debug programs which need all memory, have real-time requirements and/or are picky about vector usage.

Periscope I w/512K $599
0k memory overhead!
Periscope III $1199
hardware breakpoints and trace-back for real-time applications

C Database

C-Tree $314
highly portable file manager with full source
C-Tree / R-Tree Bundle $519
CQL ..... call
SQL access to C-Tree files
Btrieve $199
Btrieve Network $469
file manager with built-in error recovery & combined data/index files
B-Tree/ISAM $99
highly portable with full source
C-Index for Microsoft C $179
C-Index for Turbo C $89
combined data/index files with great performance and full source
pBase $179
true relational database manager

Prices subject to change without notice - ©1988 TSE All Rights Reserved • Sep/Oct 89 PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
; This is the loop that does the actual block
; blasting. Two counters are kept, and share CX by
; being separate values in CH and CL. After each
; line blast, both pointers are adjusted and the
; counters swapped, the LOOP counter decremented and
; tested and then the counters swapped again.
; ------------------------------------------------
MovEm:  MOV   CX,[bp].BWidth          ; Load atom
                                      ; counter in CH
        MOV   AH,BYTE PTR [BP].Attr ; Load attribute
                                      ; into AH
DoChar: LODSB         ; Load char from block storage
                      ; in AL
        STOSW         ; Store AX into ES:DI,
                      ; increment DI by 2
        LOOP  DoChar  ; Go back for next char if
                      ; CX > 0
```

The paragraph preceding the code is completely wrong, as is the comment on the first instruction. I read assembly language easily, yet I spent several minutes on this trivial section of code because of the confusing comments. Apparently, the programmer initially wrote and documented this section of code one way, changed the code, and forgot to update the comments.

In that last column, I also warned against documenting the obvious function of the code, and in the above fragment, every line contains some sort of obvious statement. The worst comment is "Store AX into ES:DI, increment DI by 2"—it restates what the STOSW instruction does and not what the programmer is trying to do with STOSW. Avoid the use of phrases such as "in CH" and "into AH", not only because this is obvious from the instruction, but should you change the register, you may fail to update the comment.

Below are two code snippets. The first is the rough C equivalent to the above example. If you believe the above code demonstrates good commenting conventions, surely this C code will change your mind. The second snippet demonstrates a more informative style for comments.

```
for(i=bwidth;i>0;i--) { /* load counter into i */
  c = *p++;    /* load c from block storage     */
  *q++ = c;    /* store c into *p, increment p  */
}              /* go back for next char if i>0  */
```

```
1$: mov   cx,BWidth[bp]     ; move text to display
                            ; memory
    mov   ah,bptr Attr[bp] ; use attribute from
                            ; parameter list
2$: lodsb
    stosw
    loop  2$
```

The "1$" and "2$" are OPTASM local labels, and "bptr" is an equate for the text "byte ptr". In future columns, I will cover optimization and defensive coding techniques for code sequences such as the above. Notice the potential problem with the logic in this code? (*Hint:* The C code does not suffer from it.)

MASM

Microsoft is rewriting MASM, and this time they are using a language specification. Longtime MASM users are familiar with the history of problems in this product: Source is neither upward nor backward compatible; examples in the manual do not work; bugs are documented rather than fixed (for a good laugh see page 136 of the 5.0 manual). Now, MASM has competition from OPTASM and TASM; Microsoft is concerned.

All versions of MASM through 5.1 are two-pass assemblers. "Two-pass" means that MASM looks at the source code twice during the assembly process. This is a classic design followed by most assemblers. During an initial pass, MASM parses each source line and guesses at the length of an instruction appearing on the line. MASM enters symbols found on the line into the symbol table and updates the location counter using the guessed line length. On the second pass, MASM reparses each line, and because all symbols are now defined, MASM can output the actual bytes for instructions and data.

Why does MASM generate extraneous NOPs and phase errors? The combination of a two-pass design, variable-length instructions, and uniform instruction mnemonics guarantees inefficient code generation, because MASM cannot know the correct length of all instructions on the first pass. Consider the jump instruction. Many processors offer two-byte and three-byte jump instructions. The Z80 does, but Z80 assemblers do not suffer from the NOP problem because Z80 assembly language assigns different instruction mnemonics to each flavor of jump (JR and JP). Thus, a Z80 assembler knows how long a jump instruction is on the first pass.

Intel's definition of assembly language defines JMP as the mnemonic for all classes of unconditional jumps. When the target of a jump is a forward reference and not yet defined, MASM cannot determine how long the instruction is and uses a guess of three bytes. If the guess is too long, MASM inserts a NOP; if the guess is too short, MASM generates a phase error.

So, MASM's two-pass approach to 80X86 assembly language is inadequate. What about TASM? Borland's assembler implements a one-pass assembly algorithm. The first pass performs much the same function as that in MASM, but TASM goes ahead and generates code for the instructions that are completely defined (those with no forward references).

TASM also builds a list of undefined instructions on the first pass. When the source is completely processed and the symbol table complete, TASM generates code for the instructions in that list. Unfortunately, TASM cannot change the length of an instruction and therefore has the same problem with NOPs and phase errors as MASM.

One might think Borland solved such problems with ideal mode. Wrong. Ideal mode simplifies the rather complex language syntax of MASM, resulting in programs that are easier to assemble. Who cares? I consider "rawdeal" mode an aberration; I intend to ignore this "feature."

The third member of this assembler trio is OPTASM, by SLR Systems. OPTASM implements what is known as an *n*-pass algorithm. The first pass is similar to that of a one-pass assembler except that OPTASM keeps enough information on each instruction so that on subsequent passes, should any instruction change size, OPTASM can make adjustments to appropriate instructions. OPTASM continues passes over the code until every instruction is of optimal length. Because instructions can shrink or grow, OPTASM never generates NOPs or phase errors. Even with the extra passes, OPTASM is significantly faster than either TASM or MASM.

Unfortunately for many users, reports are that the new version of MASM uses a one-pass design. Extraneous NOPs and phase errors will persist, and I find nothing new about that.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BBS

As opposed to the problem MASM and TASM have with forward references, this column has a problem with backward references. What to do when missing important information from a previous column? *PJ* might suggest that you purchase the necessary back issues—a good idea, even though I don't receive residuals on back issues. Order the back issues (and maybe a T-shirt), and then have your computer call my bulletin board system and download files pertaining to this column. The phone number is 213-559-1449. The filename for any given column is PJ*xx*.ZIP, where *xx* is the two-digit number of the volume of *Programmer's Journal* in which the column appeared. For example, the filename for this issue is PJ75.ZIP.

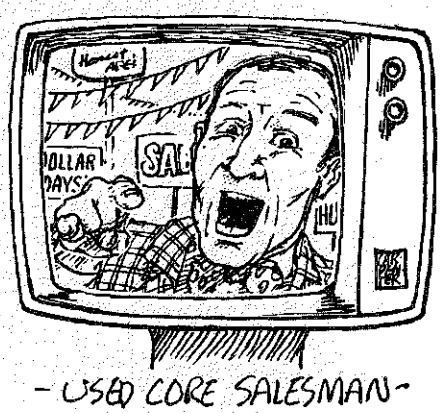
Phil Katz's shareware ZIP utility is required to unpack the component files from the archive. If you are unfamiliar with PKZIP, the self-extracting ZIP program is available from numerous sources. The most recent version is PKZ092.EXE.

That wraps up this column. Can you believe an assembly language column in which most of the listings are in C? My next column will have more information on the ideal model for assembly language, including a very efficient technique for static-variable access.

PJ

John Otken is the president of Soft Advances, where he develops debuggers for PCs and embedded systems applications. He may be reached at 10811 Washington Blvd., Suite 205, Culver City, CA 90232, 213-559-7105.

Code follows.



- USED CORE SALESMAN -

Commenting Disassembler!

SOURCER™

■ SEE HOW PROGRAMS WORK
■ EASILY MODIFY PROGRAMS

SOURCER™ creates detailed commented source code and listings from memory and executable files. Built in data analyzer and simulator resolves data across multiple segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Determines necessary assembler directives for reassembly. Includes a definition file facility to include your own remarks and descriptive labels, force data types, and more. Complete support for 8088/87 through 80286/287 and V20/V30 instruction sets. We welcome comparisons with any other product, because no product comes close to the ease of use and output clarity of SOURCER.

Sourcer is the best disassembler we've ever seen!
—PC Magazine, January 17, 1989, page 101

SAMPLE OUTPUT

Fully automatic
Program header
Assembler directives
Determines data areas and type
Detailed comments
Simulator follows segment changes
Easy to read format

```
resetprn.lst    ResetPRN v1.01              Sourcer Listing    28-Jun-89  2:34 pm      Page 1

                                PAGE  60,132
                                        RESETPRN

                                Created:   15-Apr-88
                                Version:   1.01
                                Passes:    3          Analysis Flags on: H

= 0008                          data_1e     equ     8               ; (0040:0008=378h)

                                ;--------------------------------------------------------- seg_a ----
                                seg_a       segment para public
                                            assume cs:seg_a, ds:seg_a, ss:stack_seg_b

                                resetprn    proc    far
658E:0000                       start:
658E:0000  EB 23                            jmp     short loc_1
658E:0002  52 65 73 65 74 50                db      'ResetPRN v1.01', 0Dh
658E:0008  52 4E 20 76 31 2E
658E:000E  30 31 0D
658E:0011  0040                 data_2      dw      40h
658E:0013  0D 0A 52 65 73 65    data_3      db      0Dh, 0Ah, 'Reset Printer? $'
658E:0019  74 20 50 72 69 6E
658E:001F  74 65 72 3F 20 24
658E:0025
658E:0025                       loc_1:
658E:0025  0E                               push    cs
658E:0026  1F                               pop     ds
658E:0027  BA 0013                          mov     dx,offset data_3  ; (658E:0013=0Dh)
658E:002A  B4 09                            mov     ah,9
658E:002C  CD 21                            int     21h               ; DOS Services  ah=function 09h
                                                                      ;  display char string at ds:dx
658E:002E  B4 01                            mov     ah,1
658E:0030  CD 21                            int     21h               ; DOS Services  ah=function 01h
                                                                      ;  get keybd char al, with echo
658E:0032  3C 79                            cmp     al,79h            ; 'y'
658E:0034  75 18                            jne     loc_3             ; Jump if not equal
658E:0036  8E 1E 0011                       mov     ds,data_2         ; (658E:0011=40h)
658E:003A  8B 16 0008                       mov     dx,ds:data_1e     ; (0040:0008=378h)
658E:003E  83 C2 02                         add     dx,2
658E:0041  B0 08                            mov     al,8
658E:0043  EE                               out     dx,al             ; port 37Ah, printer-2 control
                                                                      ;  al = 8, initialize printer
658E:0044  B9 8000                          mov     cx,8000h
658E:0047                       locloop_2:
658E:0047  E2 FE                            loop    locloop_2         ; Loop if cx > 0
658E:0049  B0 0C                            mov     al,0Ch
658E:004B  EE                               out     dx,al             ; port 37Ah, printer-2 control
                                                                      ;  al = 0Ch, init & strobe off
658E:004C                       loc_3:
658E:004C  B4 4C                            mov     ah,4Ch            ; 'L'
658E:004E  CD 21                            int     21h               ; DOS Services  ah=function 4Ch
                                                                      ;  terminate with al=return code
                                resetprn    endp
                                seg_a       ends

                                ;--------------------------------------------------------- stack_seg_b ----
                                stack_seg_b segment para stack
6593:0000  00C0[FF]                         db      192 dup (0FFh)
                                stack_seg_b ends

                                            end     start
```

(Source code output and inline cross reference can also be selected)

BIOS SOURCE

■ CHANGE AND ADD FEATURES
■ CLARIFY INTERFACES

for PS/2, AT, XT, PC, and Clones

The BIOS Pre-Processor™ with SOURCER provides the first means to obtain accurate legal source listings for any BIOS! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive labels such as "video_mode" and much more. Fully automatic.

SOURCER Commenting disassembler	$99.95	**UNPACKER**™ Unpack packed EXE files and more	$39.95
SOURCER with BIOS Pre-Processor	139.95	**ASMtool**™ Assembly source analyzer and flowcharter	89.95

Shipping & Handling: USA $3; Canada/Mexico $10; Other $15; CA Res. add sales tax; PS/2 trademark of IBM Corp.

All our products come with a 30 day money back satisfaction guarantee. Not copy protected. To order or receive additional information just call!

VISA **1-800-662-8266** MasterCard

V COMMUNICATIONS, INC.
3031 Tisch Way, Suite 802, Dept. J3, San Jose, CA 95128 (408) 296-4224

Number **125** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800-234-0386.
Mail-order information on pages 8 and 88.

Code from "A Library System for Assembly," by John Otken

1

LISTING 1

```
/* libgen - generate miscellaneous control files for building library

    usage: libgen *.asm

    Borland   C>tcc libgen.c wildargs.obj
    Microsoft C>cl  libgen.c setargv.obj

    You may have to specify correct path information for the
    wildargs and setargv obj files.

    John Otken, Soft Advances
*/

#include <stdio.h>

/* prototypes begin */
sort(char *(*v)[], int n);
/* prototypes end */

void main(int argc, char *argv[])
{
    char *pt;
    FILE *arf, *mak, *opt;
    int i;

    if (argc < 3) {
        printf("usage: libgen *.asm\n");
        exit(0);
    }

    arf = fopen("lib.arf", "w");
    if (!arf) {
        printf("Error opening lib.arf file\n");
        exit(1);
    }

    mak = fopen("lib.mak", "w");
    if (!mak) {
        printf("Error opening lib.mak file\n");
        exit(1);
    }

    opt = fopen("lib.opt", "w");
    if (!opt) {
        printf("Error opening lib.opt file\n");
        exit(1);
    }

    sort(&argv[1], argc-1);

    fprintf(arf, "lib.lib\ny\n");

    for( i = 1; i < argc; i++) {                    /* for each argument */
        pt = argv[i];                                /* change x.asm to x */
        for( ; *pt != '.' && *pt != '\0'; pt++);
        *pt = '\0';

        pt = argv[i];                                /*  write response file */
        fprintf(arf, "+%s &\n", pt);                 /*  & make file entries */
        fprintf(mak,"%s.obj:\t%s.asm lib.equ\n\tmasm %s/t/z;\n\n",pt,pt,pt);
        fprintf(opt," lib.lib %s.asm lib.equ\n-%s /t/z/i \\asm ;\n",pt,pt,pt);
    }

    fprintf(arf, "\nNUL\n");

/*  fprintf(mak, "lib.xxx:\tlib.lib\n\tdelete lib.lib\n\tlib @lib.arf\n"); */
    fclose(arf);
    fclose(mak);
    fclose(opt);
}

sort(char *(*v)[], int n)                    /* sorts pointer array to char */
{
    char *temp;
    int gap, i, j, k;

    for (gap = n/2; gap > 0; gap /= 2)
        for (i = gap; i <n; i++)
            for (j = i-gap; j >= 0; j -= gap) {
                k = j + gap;
                if (strcmp((*v)[j], (*v)[k]) <= 0)
                    break;
                temp = (*v)[j];
                (*v)[j] = (*v)[k];
                (*v)[k] = temp;
            }
}
```

2

LISTING 2

```
/*  asmhdr - process assembly language headers

    usage:  asmhdr document_file *.asm

    Borland   C>tcc -mc asmhdr.c wildargs.obj

    You may have to specify correct path information
    for the wildargs obj files.  Compile with compact
    memory model.  This program was written for Turbo
    C but it would be easy to modify for Microsoft C.

    John Otken, Soft Advances
*/

#include <alloc.h>      /* use malloc.h for microsoft c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LINE_SIZE       256
#define MAX_ROUTINES    1000    /* maximum routines */
#define READ_TEXT       "rt"
#define WRITE_TEXT      "wt"


/* prototypes begin */
void process_asm_file(char *fn);
void out_of_memory(void);
int stricmp_pt(struct index_tag *a, struct index_tag *b);
/* prototypes end */


struct index_tag {
    char *routine_name;
    char *routine_header;
} index[MAX_ROUTINES];


int index_cnt = 0;              /* number of index entries */
unsigned long coreused = 0;     /* total storage used */


void main(int argc, char *argv[])
{
    int i;
    FILE *fo;
```

Code continues . . .

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

epsilon
programmer's editor

State of the Art editor for DOS, OS/2, Xenix & Unix for PC/XT/AT's

Version 4.0 New!

1988 BYTE AWARD OF DISTINCTION

". . . these products–every one of them–are ones we wouldn't hesitate for a moment to recommend." (Byte, Jan. 1989)

Unrivaled Undo AND Redo

Epsilon gives you the most complete Undo and Redo of any PC programmer's editor. You can undo any sequence of changes with a touch of the Undo key. In fact, you can undo the last few hundred or so operations–you set the limit. And unlike the "full" undo offered by some other editors, Epsilon has Redo, so you can undo your undo's. You can even redo your undo's, and undo your redo's! This flexibility keeps you from losing your work by undoing too much.

Ultimate Customizability . . .

Epsilon features the most powerful extension language of any PC editor. We call it EEL, for Epsilon Extension Language. EEL gives you the rich syntax and data types of C, so you can easily extend or customize the editor to suite your taste. EEL is a real language, not an afterthought. In fact, all of Epsilon's commands are written in EEL, and you get complete source code to all the commands!

". . .but the pièce de résistance is EEL, a C-like extension language that comes with its own compiler. Lugaru Software has implemented Epsilon's entire user-interface in EEL so you can, for example, create your own syntax-sensitive modes or program the help system to browse through your source libraries." (Byte, Jan. 1989)

- C-like extension language, called EEL
- Source code to all commands
- Convenient multilevel undo plus total redo
- Scans compiler errors (customizable)
- Swaps out of memory for big compilers
- Unlimited file size, number of files
- On-line tutorial for quick startup
- As many windows as will fit on the screen
- Supports large displays (like VGA & EGA)
- Context sensitive help autoconfigures
- Tags package indexes subroutine definitions
- Files can be larger than available memory
- Fully utilizes EMS memory
- Interactively customizable keyboard
- EMACS-style command set
- Record keystroke sequences
- Multitasks compilers, even under DOS
- Language support (C, Lisp, etc.)
- Completion on file names, commands, tags, ...
- Full command line editing
- Optional backups
- Regular expression search and tagged replace
- Source level debugger & profiler for EEL
- Do interrupts from EEL (DOS)
- Call any dynamic-link library for EEL (OS/2)
- Saves deleted text (n times)
- Navigate your directory structure
- Not copy protected, for your convenience

. . . & True Multitasking Compiles!

Epsilon's exclusive multitasking facility lets you run your compilers, utilities, or even the DOS command processor, in an Epsilon window, WHILE YOU EDIT. You'll be amazed to see your compiler chugging away while you continue to edit your files. No more dead compile time! If any compiler errors appear, just touch a key and Epsilon parses the error message, then jumps right to the offending line–while your compiler is still running! Don't be fooled by misleading claims from other vendors: Epsilon is the ONLY editor for DOS with this feature! Of course, Epsilon's multitasking also works under OS/2, Xenix, and Unix.

"This feature is especially useful in a programming environment because compilers also can be run in the process buffer. . . If the screen is split into multiple windows, one of which is the process window, the user can see the compilation output while continuing to edit... A single command takes the user to the next error line in a source file and displays the error on the bottom of the screen." (PC Tech Journal, May 1987)

Ask about our free trial!

$195 direct — 60-day money-back guarantee!

Order Hotline: (412) 421-5911

FAX: (412) 421-6371

Lugaru Software Ltd. / 5843 Forbes Avenue / Pittsburgh, PA 15217

Number 271 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Otken code, continued

3

```
    if (argc < 3) {
        printf("asmhdr outfile *.asm");
        exit(0);
    }

    i = 1;
    fo = fopen(argv[i], READ_TEXT);
    if (fo != NULL) {
        printf("Error: %s exists!  Overwriting source file?\n", argv[i]);
        fclose(fo);
        exit(1);
    }

    fo = fopen(argv[i], WRITE_TEXT);
    if (fo == NULL) {
        printf("Error opening output file: %s\n", argv[i]);
        exit(1);
    }

    for(i = 2; i < argc; i++)
        process_asm_file(argv[i]);

    qsort(&index[0], index_cnt, sizeof(index[0]), stricmp_pt);

    for(i = 0; i < index_cnt; i++)
        fprintf(fo,"%s\n",index[i].routine_header);
    fclose(fo);

    printf("Used %d of %d index entries.\n", index_cnt, MAX_ROUTINES);
    printf("Used %lu bytes.\n", coreused);
                /* microsoft c does not have a coreleft function */
    printf("%lu free bytes.\n", (unsigned long) coreleft());
}



void process_asm_file(char *fn)
{
    char *p, *q;

    char line[LINE_SIZE],    /* file line buffer */
         buf[3000];          /* temp buffer */

    FILE *fi;

    fi = fopen(fn, READ_TEXT);
    if (fi == NULL) {
        printf("Error opening input file: %s\n", fn);
        exit(1);
    }

    while (fgets(line, LINE_SIZE, fi) != NULL) {
        if (line[0] == ';'  &&  line[1] == ';'  &&  line[2] == '\t') {

            p = buf;
            do {
                p = stpcpy(p, line);
            } while (fgets(line, LINE_SIZE, fi) != NULL  &&  line[0] == ';');
            if (*(p-1) == '\n'  &&  *(p-2) == ';')  p -= 2;
            *p = '\0';
            if ((index[index_cnt].routine_header = strdup(buf)) == NULL)
                out_of_memory();
            coreused += strlen(buf)+1;

            p = &buf[3];         /* extract routine name from header */
            q = buf;
            while (*p != '\n') {
                if (*p == ' ')  *p = '_';
                *q++ = *p++;
            }
            *q = '\0';

            if ((index[index_cnt++].routine_name = strdup(buf)) == NULL)
                out_of_memory();
```

4

```
            coreused += strlen(buf)+1;
        }
    }
    fclose(fi);
}



void out_of_memory(void)
{
    printf("Out of memory");
    exit(1);
}



int stricmp_pt(struct index_tag *a, struct index_tag *b)
{
    return stricmp(a->routine_name, b->routine_name);
}
```

LISTING 3

```
        title   outchr
        .model  small
        .code
        extrn   ms_dos:near
        public  outchr

;;      outchr
;
;       entry   AL      char
;       uses    AX
;
outchr  proc
        push    ax
        push    dx
        mov     ah,2
        mov     dl,al
        call    ms_dos
        pop     dx
        pop     ax
        ret
outchr  endp

            end
```

LISTING 4

```
        title   ms_dos
        .model  small
        .code
        public  ms_dos

;;      ms_dos
;
;       note    entry/exit conditions defined
;               in dos technical reference
;
ms_dos  proc
        int     21h
        ret
ms_dos  endp

            end
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROGRAMMER'S PRODUCTIVITY PACK

"I Love It!"

"I'll use this all the time"
Gary Gruber-Data Base Designs

"...best product in it's class... I wouldn't code a serious project without it!" *Barry Butler-Powerline Software*

Here are just a few of the features –

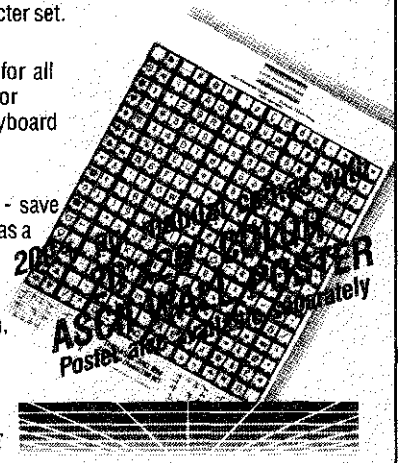
PROGRAMMER'S CALCULATOR
Decimal, Binary, Octal and Hexadecimal simultaneously - Signed/Unsigned modes - full 32 bit values. SHL/SHR,SAL/SAR,ROL/ROR,RCL/RCR, - AND, NOT, OR, XOR, XCHG.

ASCII & COLOR ATTRIBUTE CHART
Displays and describes color and monochrome attributes - Enlarged display of box drawing and graphics characters - Entire ASCII and IBM extended character set.

KEYSTROKE REFERENCE CENTER
BIOS Scan Codes and dBase INKEY ( ) values for all keys and key combinations - Portability Notes for keyboard differences - Live display of the BIOS Keyboard Status Bytes.

KEYSTROKE RECORDING/PLAYBACK
Up to 100 recordings of 1,000 Keystrokes each - save recordings to disk - Grab text from screen for use as a Keystroke Recording.

So easy you won't even read the manual!



PLUS THESE EXTRAS!

Screen capture with filtering, background printing, screen blanking with hard disk parking, memory-resident (MS-DOS) & STAND-ALONE (DESQview, Omniview) versions included.

30 DAY MONEY BACK GUARANTEE

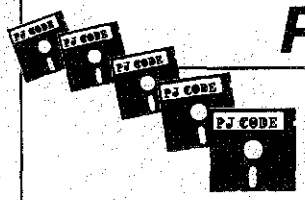
ONLY $79.95 plus shipping and handling

CALL FOR FREE BROCHURE 800-326-5615

FALK DATA SYSTEMS
5322 Rockwood Ct. El Paso, Texas 79932

Number **267** on the Reader Service Card

PJ Code Is Available!



On disk from:

Programmer's Journal
P. O. Box 30160
Eugene, OR 97403
800-234-0386
503-747-0800 in Oregon
All Code Disks $10.00 each.
To order, see ad on page 88.
PJ Code DiskScriptions are available for only $39.95.
To order, see ad on page 8.

On BBS from:

FIDO/OPUS Network
Blazer Net 152 Node 9
503-343-9724

Generation Five BBS
(Voice) 301-495-6690
(Data) 301-495-2932
Accessible via PC Pursuit, Washington D.C. From Files submenu, select "P" for Personal Downloads.

Gilmore Systems of Southern California
5447 Indian Hills Drive
Simi Valley, CA 93063
(Voice) 805-582-1360
(Data) 805-582-9306

Ithaca Street BBS
1145 Ithaca
Boulder, CO 80303
(Voice) 303-494-8865
(Data) 303-494-8868

PCS-49 on CompuServe
Section 9
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
(Voice) 800-MISOSYS

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention *PJ* when you call.

Multitasking

for the IBM-PC/XT/AT
with

Multidos Plus

Ideal for developing process control, data acquisition, communications, and other applications which require a multitasking environment. Check these features which make Multidos Plus an unbeatable value:

- Run up to 32 programs concurrently
- Your software continues to run under PC-DOS or MS-DOS. No need to learn a new operating system.
- Use the compilers you already have. Supports software written in most compiled languages or assembly language.
- Operator commands to load and run programs, change priority, check program status, and to abort/suspend/resume task execution.
- Optional independent foreground and background displays.
- *Programmatic interface through* INT 15H for the following:
  - * Intertask messaging to send, receive, check or wait on 63 different message queues.
  - * Task control by means of semaphores. Get, release, check, or wait on semaphores.
  - * Change task priority.
  - * Suspend task for specified interval.
  - * Spawn and terminate external and internal tasks.
  - * Disable/enable multitasking.

Requires: IBM PC/XT/AT or true clone. MS-DOS (or PC-DOS) 2.0 or later. Enough memory to hold Multidos Plus (64KB) and all your application programs. You may also need 4 or 16KB of memory for "hidden screens" for each active task.

$24.95 or

$99.95 with source code (written in Microsoft QuickC 2.00 and Assembler)

To order call (800) 678-2141. FAX (508) 655-8860. For technical information and foreign orders call (508) 651-0091. Visa and Mastercard accepted. Or send check or money order (Drawn on U.S. banks only) to:

Nanosoft

13 Westfield Road
Natick, MA 01760

Add $2.00 for shipping & handling.
Foreign orders add $5.00 for shipping & handling.
COD orders add additional $3.00
Massachusetts shipments add 5% sales tax.

Number **109** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The World in Your Hands

by
Gene Fowler

Turbo Pascal 5.0's graphics primitives are a first-rate tool . . . but there's no world for scaling or 3-D. With GRAPHWLD.TPU, that changes.

GRAPHWLD.TPU isn't a "shell" or "interface" creating the illusion that you are working with a kernel like that in Graphix Toolbox but with BGI's stroked fonts and surface fills inside. It holds the same primitives or raw building blocks that BGI supplies in GRAPH.TPU, but with add-on "**wld2vport**" translators wrapped around each.

The names of these "parallel" drawing primitives identify the "originals" they call. If you are working in a world and would call **Line(vpx1,vpy1,vpx2,vpy2)**, you instead call **WLine(wx1,wy1,wx2,wy2)**, and **WLine** translates the world coords to viewport coords and calls **Line(vpx1,vpy1, vpx2,vpy2)**.

Inventory

The world-handling primitives in the unit are **WPutPixel** and **WGetPixel**; **WLine**, **WLineTo**, and **WLineRel**; and **WMoveTo** and **WMoveRel**. Three stand-alone translators allow you to translate coordinates (**w2vp**), distances or relative coordinates (**w2vpRel**), and radius lengths (**w2vpRadius**).

In some instances, you can translate a set of display coordinates once, hang on to the *vp* values, and use them in repeated calls. In other instances, you will use procedures from GRAPH.TPU involving a number of params that a "parallel" package would have to take time to relay. In the following example, **w2vp** is used to translate a pair of world coordinates into viewport coordinates to place text in the world picture plane using **OutTextXY**:

```
SetTextstyle(DefaultFont,HorizDir,1);
wx := 200; wy := 250;
w2vp(wx,wy,vpx,vpy); (uses stand-alone translator)
OutTextXY(vpx,vpy,'press any key...')
```

We need a *vpx* and a *vpy* for **OutTextXY**. A package would have to relay the text through a parameter and a stuff-in inside. It's quicker, then, to use the stand-alone **w2vp** and call **OutTextXY** from the program.

In the following example, you see a use for **w2vpRadius**:

```
wt1 := (MaxWX + 1) shr 1; {wx for center point}
wt2 := (MaxWY + 1) shr 1; {wy for center point}
w2vp(wt1,wt2,vpt1,vpt2);
wRadius := 50; wAspRatio := 0.44;
w2vpRadius(wRadius, vpRadius,wAspRatio;
circle(vpt1,vpt2,vpRadius);
```

Here, **w2vpRadius** takes VAR params *wRadius* and *vpRadius* and translates. It takes *wAspRatio* value. (Demos to run on all cards are in 640:200 world.) To call Circle, you must give the coordinates of the center and a radius length. To draw in your world, use **w2vpRadius** to translate. You need VAR params *wRadius* and *vpRadius*, and you must supply the world's aspect ratio (not the screen's) in the VALUE param *wAspRatio*.

"Birthing" a World

The inventoried primitives won't work if you haven't created a world for them to work in. GRAPHWLD.TPU's main procedure, therefore, is **CreateWorld(ULx,ULy,LRx,LRy)**. The params are the minimum and maximum *x* and *y* world values, not the viewport values. **CreateWorld** sets up all the reference values for doing the translations and some other related references for the procedures inventoried above to use.

In TP5.0 BGI, all coordinates are viewport relative; so, it is *very* important that you set your viewport *before* you call **CreateWorld**. And if you set a new viewport and are going to use the same world (for scaling), you must repeat your **CreateWorld**, because **CreateWorld** uses the viewport information to set up for translation.

Doing a Cartesian Flip

You can flip your world "upside down" to have the Cartesian coordinate field used in science and business. This is a *y*-axis flip, and you accomplish it by having *ULy* > *LRy*. In **CreateWorld**, these will be reversed, and translation will treat of a "right-side-up" world; then, at the end, the *vpy* will be set at the distance from the lower screen edge that it was from the higher screen edge. The trigger for this is the *ULy* > *LRy*.

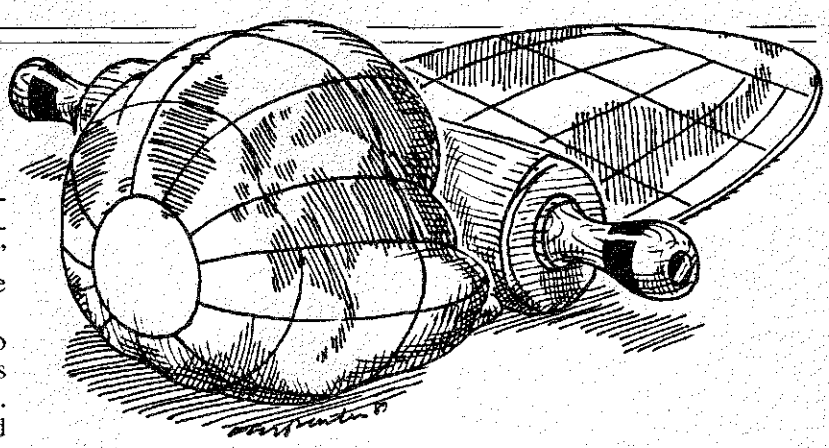
A Demo Worth Doing

If you are, like me, a "gentleperson amateur" or even (and it happens) a "Sunday programmer," you'll have fun with this demo.

I've bypassed the expected demo that sets up a world "differently proportioned" than your usual viewport world to show off my procedures and the power to scale. I'm going to step off into a wonderful three-dimensional world for rotating and moving about "solid" geometrical objects.

GRWDEMO.PAS is definitely "stripped down." For starters, there's no error handling; so type carefully. The object is a cube—or it *was*, for only two faces remain. A single face was difficult to see as rotating rather than distorting.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



There is no hidden surface marking and no surface filling, and so you have a "wire frame" figure. There's no flag system to avoid "double" drawing of lines; I just leave off one line on one of the surfaces.

When you run the program, you get an intro screen that tells you what's going on and asks for three rotation params for yaw, roll, and pitch. You must enter each in degrees (0–360), and there's no error prevention or handling. The program converts to radians for the math. You don't enter translations, picture-plane distance, object distance, or other params. Check in **InitWorld** to find those things, and add them to the data entry (**GetParams**) if you wish. As you enter the third param, the program draws the figure (see the illustration). Press a key, and you can repeat or quit. You can, of course, write in code to have the repeats "adjust" angles. As it is, you must enter your three params again.

It's crude, but the "how to" of 3-D object handling is clearly set out. Start reading the main program, and go look at the procedures in the order they're called. It will read clearly—the strength of Pascal identifiers.

Finagling a World

Engineering texts stretch my faculties a bit painfully; so, I extracted the maths from Lee Adams' *High-Performance Interactive Graphics (for IBM PCs and Compatibles)* (Tab Books, 1987). And a recommended world for 3-D graphics came along with the maths.

It's a 4:3 world, or, in units, 800:600. You'd enter that (in **InitWorld**):

```
CreateWorld(-399,-299,400,300)
```

Note the negative (minimum) coordinates—"-1" because the 0,0 coordinates specify one of the 800x600 locations. This is the world I've typed into GRWDEMO.PAS as it's printed.

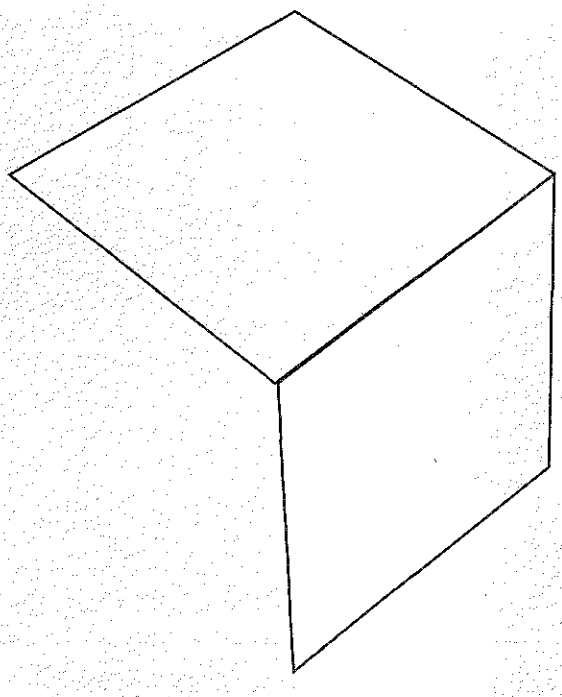
Adams gets this "aspect ratio" (built into the world) from the display space on most personal computer monitors and all TV monitors. Most discussions of aspect ratio have to do with pixel shape, not display-area shape. This presented no problems to Adams, but it will to us. Adams assumed CGA, EGA, VGA, PCjr system, a Tandy clone to the latter, and the monitors attached to those. But more important, he assumed programming in the BASICA interpreter and equivalents. He mentions BASICA's card "drivers," and my guess is that those drivers, for those cards, take care of the pixel-shape aspect ratios. And monitors attached to them have 4:3 display areas.

My HGC+–driven 1985 IBM monochrome monitor did not give me squares "on the picture plane" in this 4:3 world, and so I had to finagle a workable world. Not only that, but when I decided to fire up my **GraphPrt** and hardcopy my screen (not viewport), I had to finagle a different world to get the squares on paper instead of on screen.

I have an 800:550 world for my screen and an 800:640 for my screen2printer. Those are "plastic ruler"–calibrated squares. I've been unable to find any functional relation between that extra "contraction" (and later overcompensating "expansion") and the 3:2 (9-inch-by-6-inch) display space on my monitor, the pixel-shape aspect ratio of 0.75 BGI assigns to the HGC, or the 0.725 that works better on my monitor. So I can't give you a nice **GetAspectRatio**-fed way to compute your "finagle." But I have set up GRWDEMO to make finagling a world pretty easy.

Run GRWDEMO in the IDE. Put the cursor on the **CreateWorld** line in **InitWorld** and (since you'll be adjusting both each time) on one of the two *y* coordinates.

Ctrl-F9 runs it. Enter three zeros. It'll draw one of the faces on the picture plane, with the other bent back at 90 degrees. Apply a stiff ruler (the screen may be curved) and measure. Press any key and then Q (for "quit") to return to code. Adjust the pair of *y* coordinates. If your "square" was too tall, decrement the *y* coordinates; if it was too flat, increment them. A few run-throughs and you should have your world.



Printout of a GRWDemo screen with 45-degree yaw, 0-degree roll, and -45-degree pitch. With other angles, you might see a tangle of wire framings, an incentive to "blank" the faces as you draw them.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

And "Flipping" a World

When you have *your* world in the **CreateWorld** line, look at the commented-out **CreateWorld** line below it: It has the y coordinates reversed. Comment out the world you've been using and remove the commenters from this one, run the program, enter your same three zeros, and see what happens to the drawing and text.

Nailing Together an Object

You can use what I've set out as GRWDEMO as a kind of "testbed" or "breadboard" and build up whatever piece of furniture you'd like, learning how it works by having built it. For instance, you can go into **ConstructModel** and "weld on" the other four faces of the cube.

First, of course, you've got to go into **InitWorld** and add two vertices to **WObj**. That means you have to go up to your types and get all those arrays expanded to hold those additional coordinate sets. And you'll have to go where view and display coordinates are calculated and check loop counts so that they all get done. And later, when you sheepishly go back and find one more loop or declaration (thanking your stars for TP5's integrated debugger), you'll figure you've learned something about all this.

Then, in **ConstructModel**, you can fake up those sides like I did my second one. Just draw lines not already drawn when you add a face. You'll get your cube quick and easy. And that's fine for wire frame. Later, you'll want to do solid objects or engineer's X-ray (with special line style). At that time, you're going to have to bring in the "hidden surface" calculation to know what kind of lines to draw for a given face on your rotated cube. When you're ready for that, I recommend Appendix B in Adams' book. Even before that, you can change the way you draw your wire-frame cube to get ready for the more sophisticated handling.

You will need a stack of flags for the twelve lines so that a set flag says, "Draw this line." Then, you have a single figure drawing chunk of code. Each face handler sets the flags of those lines it requires. In your wire frame, each flag gets set, twice; there is a face on each side of a line.

Later, when you are set to do X-ray and solid figures, you will have two stacks of flags—one for the full lines for the "visible" surfaces and one for the "invisible" surfaces (controlling the drawing of dashed or unseen lines). Draw the full lines first, and when drawing a full line, turn off the corresponding dashed or empty line in the other stack.

When you begin reshaping the code into that sort of "designed" elegance, you leave the realm of hacking results onto the screen for the software engineer's workshop. But I'll tell you, I've been poking around in here (as you can see), and I haven't yet run into a "No Trespassing" sign or a locked cabinet, and nobody's glared at me. And I'm having a ball. So now you understand why the demo wasn't a quiet little testbed where I could run through the GRAPHWLD procedures one by one. PJ

With credentials only as outlaw poet, Gene Fowler enters computing as a "gentleperson amateur" programmer with a philosophy of "jump in and work out a way to do it before coming across documentation that it's not possible." You can write Gene by non–e-mail at 1432 Spruce Street, Berkeley, CA 94709.

Code follows.

TechnoJock's Turbo Toolkit

Attention Turbo Pascal programmers.....

Cut your program development time in half and create sophisticated user interfaces in minutes with *TechnoJock's Turbo Toolkit!*

The *Toolkit* contains twelve units with all source code included, and a 300 page manual full of examples. These range from a comprehensive unit that provides a Sidekick Plus style directory window with all the expected sorting features, to a basic procedure that writes text centered on the screen.

And it's easy to use! If you've bought other toolkits but didn't use them because of all their complicated rules ... TTT is for you.

TECHNOJOCK SOFTWARE

- ultra fast screen writing
- EGA/VGA extended lines
- exploding boxes
- virtual screens
- sorted directory listings
- comprehensive form input
- pulldown menus
- nested menus
- string manipulation
- windows
- full mouse support
- keyboard control
- date manipulation

"TechnoJock's Turbo Toolkit may just cut your work in half! It provides easy-to-implement procedures.... designed for all Turbo Pascal programmers. With this Toolkit everyone becomes a TechnoJock."
Turbo Users Group

"Here, despite the whimsical name, is shareware at its best: an innovative product, a superb manual, quick mail-order service, and a reasonable price... If you're programming in Turbo Pascal, you absolutely must have TTT!"
Public-Domain Software & Shareware 2nd edition

"Looks great! We have decided to implement your Toolkit in in-house programs for Tech Support."
Rich Kubota
Technical Support
Borland International

TechnoJock Software, Inc.
PO Box 820927
Houston, TX 77282-0927

Orders: 1-800-242-4775
Enquiries: (713) 493-6354

Only $49.95!

Plus $5.00 shipping and handling (Canada $10.00, overseas $15.00). Texas residents add 8% Sales Tax.

Send checks to TechnoJock Software. Send Visa/Mastercard details to: Public (Software) Library PO Box 35705 Houston TX 77232-5705

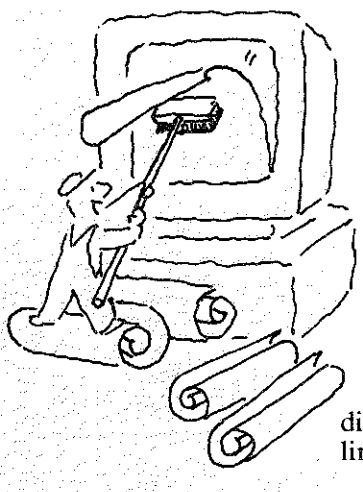
Number 289 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Two great tools.

NEW! SUPPORTS TURBO PASCAL 4.0, 5.0 & 5.5 AND QUICKPASCAL!

SAYWHAT?! The lightning-fast screen generator.



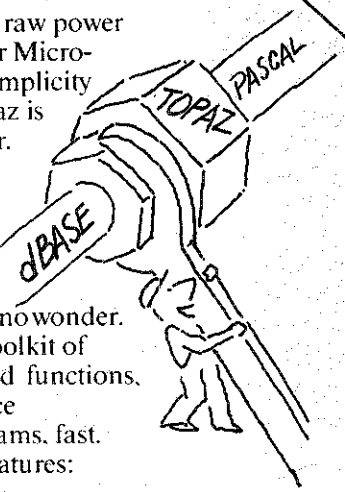
Whether you're a novice programmer longing for simplicity, or a seasoned pro searching for higher productivity, you owe it to yourself to check out Saywhat. You see, with Saywhat, you can build beautiful, elaborate, color-coded screens in minutes! That's right. Truly *fantastic* screens for menus, data entry, data display, and help-panels that can be displayed with as little as one line of code in *any* language.

Here's what you get:

- Design screens, windows, and *moving bar menus!*
- Easy-to-use, powerful editor lets you create screens in a jiffy.
- Pop up your screens and menus with one line of code in dBASE, all the dBASE compilers, your favorite BASIC, Pascal, *or any other language!*
- Screen Library Manager.
- Generates runtime code.
- No runtime license or royalty fees.
- Comes with a 100 page manual, plus dozens of sample programs and free utilities.

$49.95

TOPAZ The breakthrough DBMS toolkit for Pascal



If you'd like to combine the raw power and speed of Turbo Pascal or Microsoft's QuickPascal with the simplicity and elegance of dBASE, Topaz is just what you're looking for. That's because Topaz was specially created to let you enjoy the best of *both* worlds. The result? You create complete, truly dazzling applications in a very short time. And no wonder. Topaz is a comprehensive toolkit of dBASE-like commands and functions, designed to help you produce outstanding, polished programs, fast. Check out these powerful features:

- Over 200 routines all with easy-to-use, dBASE-like syntax.
- Data entry routines like SAY, GET, PICTURE, RANGE, color selection, unlimited data validation.
- Open up to 10 DBF files, with up to 7 indexes with USE, SELECT, SKIP, APPEND, PACK, INDEX ON, SET INDEX TO, and FIND.
- No need to buy dBASE. CREATE, BROWSE and REPORT utilities included.
- Easily implement Saywhat and Lotus-style moving bar menus.
- BROWSE any DBF file with just one line of code! Programmable and windowed too.
- Pick from windowed data or filenames with one line of code.
- Comprehensive Time & Date math in 7 international formats.
- Powerful code and report generators included!
- Comes with a complete 250 page manual, plus sample programs to get you started.

$74.95

ORDER NOW. YOU RISK NOTHING.

Thousands of satisfied customers have already ordered from us. Why not visit your dealer or call toll-free, right now and put Saywhat and Topaz to the test yourself? They're fully guaranteed. You don't risk a penny.

Special limited-time offer! Save $26. Buy Saywhat?! and Topaz together for just **$99** (plus $5 shipping and handling).

Visit your nearest dealer or call toll-free:
800-468-9273
In California: 800-231-7849
International: 415-571-5019

Software Science, Inc.
The Research Group
100 Valley Drive, Brisbane, CA 94005

MONEY BACK GUARANTEE.
If you aren't completely delighted with Saywhat or Topaz, for any reason, return them within 30 days for a prompt, friendly refund.

Dealers: SAYWHAT?! and TOPAZ are available from Kenfil Distribution, and in Europe from ComFood Software, W. Germany 49-2534-7093

Guaranteed!

SOFTWARE SCIENCE INC

Number 198 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800-234-0386.
Mail-order information on pages 8 and 88.

Code from "The World in Your Hands," by Gene Fowler

1

LISTING 1

```
{GraphWld.tpu Copyright (C) 1989 by Gene Fowler

GraphWld.tpu extends Graph.tpu to handle world co-
ordinates by providing parallel drawing procedures
that translate world to viewpoint coordinates and
call the original procedures from Graph. There are
also two stand-alone translators (one for x,y co-
ordinates and one for dx,dy relative coordinates
or distances) when one translation yields variables
for repeated calls or a parallel procedure would
have to relay extra params.

The "central" procedure is CreateWorld(ULx,ULr,LRx,
LRy). The params define your world. This procedure
is called AFTER any defining of a viewport in which
the world will exist and BEFORE any use of the
translating procedures.
}
unit GraphWld; {world coordinates ext. to Graph.tpu}

interface

uses crt, graph;

procedure CreateWorld(ULx, ULy, LRx, LRy : real);

procedure w2vp(Var wx, wy : real; var vpx, vpy : integer);

procedure w2vpRel(Var wdx, wdy : real; var vpdx, vpdy : integer);

Procedure w2vpRadius(var wRadius: real; var vpRadius: word; wAspRatio: real);

procedure WPutPixel(wx, wy : real);

function  WGetPixel(wx, wy : real) : word;

procedure WLine(wx1, wy1, wx2, wy2 : real);

procedure WRectangle(wx1, wy1, wx2, wy2 : real);

procedure WLineTo(wx, wy : real);

procedure WMoveTo(wx, wy : real);

procedure WLineRel(wdx, wdy : real);

procedure WMoveRel(wdx, wdy : real);

implementation

var
   xv,yv, x1v,y1v,x2v,y2v : integer;
   MaxColor : word;
   RatioX, RatioY : real;
   VPMaxX, VPMaxY : integer;
   ViewP : ViewPortType;
   WXTotal, WXNegAdj, WYTotal, WYNegAdj,
   FTemp : real;
   FlipYFlag : boolean;
   ivpdx, ivpdy : real;
   xAsp, yAsp   : word;
   vpAspRatio : real;

procedure CreateWorld{(ULx, ULy, LRx, LRy : real)};

begin
   GetViewSettings(ViewP);  {Viewport must be set before world}
   with ViewP do
     begin
```

2

```
       VPMaxX := x2-x1;
       VPMaxY := y2-y1;
     end;
   if ULy > LRy then  { for Cartesian flip; corresponding vpy adjust. in
                        the procedures. }
     begin
       FlipYFlag  := true;
       FTemp      := ULy;
       ULy        := LRy;
       LRy        := FTemp;
     end
   else FlipYFlag := false;
   WXTotal := LRx - ULx + 1;
   if ULx < 0 then WXNegAdj  := Abs(ULx) else WXNegAdj := 0;
   WYTotal := abs(LRy - ULy) + 1;
   if ULy < 0 then WYNegAdj  := Abs(ULy) else WYNegAdj := 0;
end;

procedure w2vp{(Var wx, wy : real; var vpx, vpy : integer)};

begin
   RatioX := (VPMaxX + 1) / WXTotal; Ratio Y := (VPMaxY + 1) / WYTotal;
   wx := wx + WXNegAdj;
   wy := wy + WYNegAdj;
   vpx := round(wx * RatioX); vpy := round(wy * RatioY);
   if FlipFlag then vpy := VPMaxY - vpy;
end;

procedure w2vpRel{(Var wdx, wdy : real; var vpdx, vpdy : integer)};

var
   NFlagX : boolean;
   NFlagY : boolean;

begin
   RatioX := (VPMaxX + 1) / WXTotal; RatioY := (VPMaxY + 1) / WYTotal;
   if wdx < 0 then NFlagX := True else NFlagX := False;
   if not FlipYFlag then
     if wdy < 0 then NFlagY := True else NFlagY := False;
   else
     if wdy < 0 then NFlagY := False else NFlagY := True;
   wdx := abs(wdx); wdy := abs(wdy);
   vpdx := round(wdx * RatioX); vpdy := round(wdy * RatioY);
   if NFlagX then vpdx := -vpdx;
   if NFlagY then vpdy := -vpdy;
end;

Procedure w2vpRadius{(var wRadius: real; var vpRadius: word; wAspRatio: real)};

var
   wdx, wdy : real;

begin
   RatioX := (VPMaxX + 1) / WXTotal; RatioY := (VPMaxY + 1) / WYTotal;
   wdx    := sqrt(sqr(wRadius) / (1 + (sqr(wAspRatio))));
   wdy    := wAspRatio * wdx;
   ivpdx  := (wdx * RatioX);
   GetAspectRatio(xAsp, yAsp);
   vpAspRatio := xAsp / yAsp;
   ivpdy := (wdy * RatioY) * (wAspRatio / vpAspRatio);
   vpRadius := round(sqrt(sqr(ivpdx) + sqr(ivpdy)));
end;

procedure WPutPixel{(wx, wy : real)};

begin

   wx := wx + WXNegAdj;
   wy := wy + WYNegAdj;
   RatioX := (VPMaxX + 1) / WXTotal; RatioY := (VPMaxY + 1) / WYTotal;
   xv := round(wx * RatioX); yv := round(wy * RatioY);
   if FlipYFlag then yv := VPMaxY - yv;
   MaxColor := GetMaxColor;
   PutPixel(xv, yv, MaxColor);
end;

function WGetPixel{(wx, wy : real) : word};
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
begin
  wx := wx + WXNegAdj;
  wy := wy + WYNegAdj;
  RatioX := (VPMaxX + 1) / WXTotal; RatioY := (VPMaxY + 1) / WYTotal;
  xv := round(wx * RatioX); yv := round(wy * RatioY);
  if FlipYFlag then yv := VPMaxY - yv;
  WGetPixel := GetPixel(xv, yv);
end;

procedure WLine(wx1, wy1, wx2, wy2 : real));

begin
  RatioX := (VPMaxX + 1) / WXTotal; RatioY := (VPMaxY + 1) / WYTotal;
  wx1 := wx1 + WXNegAdj;
  wy1 := wy1 + WYNegAdj;
  wx2 := wx2 + WXNegAdj;
  wy2 := wy2 + WYNegAdj;
  x1v := round(wx1 * RatioX); y1v := round(wy1 * RatioY);
  x2v := round(wx2 * RatioX); y2v := round(wy2 * RatioY);
  if FlipYFlag then
    begin
      y1v := VPMaxY - y1v;
      y2v := VPMaxY - y2v;
    end;
  Line(x1v,y1v,x2v,y2v);
end; {WLine}

procedure WRectangle(wx1, wy1, wx2, wy2 : real));

begin
  RatioX := (VPMaxX + 1) / WXTotal; RatioY := (VPMaxY + 1) / WYTotal;
  wx1 := wx1 + WXNegAdj;
  wy1 := wy1 + WYNegAdj;
  wx2 := wx2 + WXNegAdj;
  wy2 := wy2 + WYNegAdj;
  x1v := round(wx1 * RatioX); y1v := round(wy1 * RatioY);
  x2v := round(wx2 * RatioX); y2v := round(wy2 * RatioY);
  if FlipYFlag then
    begin
      y1v := VPMaxY - y1v;
      y2v := VPMaxY - y2v;
    end;
  Rectangle(x1v,y1v,x2v,y2v);
end; {WRectangle}

procedure WLineTo(wx, wy : real));

begin
  RatioX := (VPMax + 1) / WXTotal; RatioY := (VPMaxY + 1) / WYTotal;
  wx := wx + WXNegAdj;
  wy := wy + WYNegAdj;
  xv := round(wx * RatioX); yv := round(wy * RatioY);
  if FlipYFlag then yv := VPMaxY - yv;
  LineTo(xv,yv);
end; {WLineTo}

procedure WMoveTo(wx, wy : real));

begin
  RatioX := (VPMaxX + 1) / WXTotal; Ratio Y := (VPMaxY + 1) / WYTotal;
  wx := wx + WXNegAdj;
  wy := wy + WYNegAdj;
  xv := round(wx * RatioX); yv := round(wy * RatioY);
  if FlipYFlag then yv := VPMaxY - yv;
  MoveTo(xv,yv);
end; {WMoveTo}

procedure WLineRel(wdx, wdy : real));

var
  NFlagX : boolean;
  NFlagY : boolean;

begin
  RatioX := (VPMaxX + 1) / WXTotal; Ratio Y := (VPMax + 1) / WYTotal;
  if wdx < 0 then NFlagX := True else NFlagX := False;
  if not FlipYFlag then
    if wdy < 0 then NFlagY := True else NFlagY := False
```

Code continues . . .

Turbo Pascal Tools to Help You Program Faster and Easier

A Multi-User B-Tree Toolkit

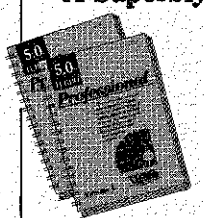
Write powerful network compatible databases faster and easier using **B-Tree Filer 5.0**. You'll have the fastest, safest, most flexible databases – no rigid structure, no TSR hassles, no running out of files. And they're compatible with Novell, 3Com, MS-NET, and others.

You get: ■ Fixed and variable length records ■ Two billion records per data base ■ Up to 100 indexes per file ■ Flexible record locking ■ Fail-safe mode with journaling ■ Units for sorting, browsing, reindexing, and network control.

B-Tree Filer includes full source code, documentation, technical support, and you pay NO royalties.

B-Tree Filer 5.0 is only $125. (single user)
With network support, only $175.

"A Superbly Crafted Toolbox"



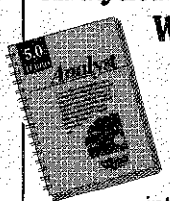
Turbo Professional 5.0 is a library of more than 600 state-of-the-art routines optimized for Turbo Pascal 5.0. It includes complete source code, comprehensive documentation, and powerful and useful demo programs.

You get: ■ TSR management ■ Menus and windows ■ BCD ■ Large arrays and more. *New Routines!* Complete mouse support ■ Window-oriented text editor ■ Scrolling data entry screens ■ Versatile pick lists ■ On-line reference guide, and more.

"Turbo Professional...a superbly crafted toolbox... [makes] the hard stuff so easy."
Kent Porter, Dr. Dobb's Journal, 4/88

Turbo Professional 5.0 is only $125.

Analytical Tools to Help You Write Better Programs



You'll write better Turbo Pascal 5.0 programs easier and faster using the powerful analytical tools of **Turbo Analyst 5.0**. You get an integrated development environment with: ■ File Browser ■ Mouse support ■ Hotkey access to the compiler, debugger and the analytical tools.

The tools include: Pascal formatter, Program structure analyzer, Program lister, three Execution profilers, and Unit disassembler.

"Turbo Analyst...a valuable tool for every Turbo Pascal programmer. The Program Structure Analyzer...a timesaver. The integrated environment...impressive."
Namir Shammas, Turbo Tech Report, 7/88

Turbo Analyst 5.0 is only $99.

Satisfaction guaranteed or your money back within 30 days. Turbo Pascal 4.0 or 5.0 is required. Shipping and taxes prepaid in U.S. and Canada. Elsewhere add $15 per unit. For more information call (408) 438-8608.

TURBO Power

TurboPower Software P.O. Box 66747 Scotts Valley, CA 95066-0747

Call toll-free for credit card orders.
1-800-333-4160
Continental U.S. and Canada

Number 25 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Fowler code, continued

4

```
    else
      if wdy < 0 then NFlagY := False else NFlagY := True;
    wdx := abs(wdx); wdy := abs(wdy);
    xv := round(wdx * RatioX); yv := round(wdy * RatioY);
    if NFlagX then xv := -xv;
    if NFlagY then yv := -yv;
    LineRel(xv,yv);
end;  {WLineRel}

procedure WMoveRel{(wdx, wdy : real)};

var
    NFlagX : boolean;
    NFlagY : boolean;

begin
    RatioX := (VPMaxX + 1) / WXTotal; RatioY := (VPMaxY + 1) / WYTotal;
    if wdx < 0 then NFlagX := True else NFlagX := False;
    if not FlipYFlag then
      if wdy < 0 then NFlagY := True else NFlagY := False;
    else
      if wdy < 0 then NFlagY := False else NFlagY := True;
    wdx := abs(wdx); wdy := abs(wdy);
    xv := round(wdx * RatioX); yv := round(wdy * RatioY);
    if NFlagX then xv := -xv;
    if NFlagY then yv := -yv;
    MoveRel(xv,yv);
end;  {WMoveRel}

end.
```

LISTING 2

```
{GRWDEMO.PAS Copyright (C) 1989 by Gene Fowler
GRWDemo.pas is a stripped-down 3-D object
rotator and translator to be compiled in Turbo
Pascal 5.0 using crt, graph, and graphWld. It
uses three GraphWld procedures: CreateWorld in
InitWorld; WLine in ConstructModel; and w2vp
in WritePressKey.
}
program GraphWldDemo;

uses crt, graph, graphwld;

type
    ObjectVertex = record
                     x, y, z : real
                   end;
    WorldObj     = Array[0..5] of ObjectVertex;     {world coords}
    ViewObj      = Array[0..5] of ObjectVertex;     {view coords }
    ScreenVertex = record
                     sx, sy : Real
                   end;
    DisplayObj   = Array[0..5] of ScreenVertex;     {display coords}

var
    {BGI Init and other control variables}
    gdriver, gmode, ecode : integer;
    MaxX, MaxY : integer;
    MaxColor : word;
    ViewP : ViewPortType;
    GoAgain : char;

    {The Array variables}
    WObj   : WorldObj;
    VObj   : ViewObj;
    DObj   : DisplayObj;

    {3-D drawing variables}
    Dist : Real;
    YawDeg, RollDeg, PitchDeg,
    YawRad, RollRad, PitchRad,
    SinYawRad, CosYawRad, SinRollRad,
    CosRollRad, SinPitchRad, CosPitchRad,
    TransX, TransY, TransZ : Real;
```

5

```
    {work variables for calculations}
    x, y, z, xa, ya, za,
    x1, x2, x3, x4, y1, y2, y3, y4,
    z1, z2, z3, z4,
    sx, sy : Real;
    i : byte;

    {VAR params for the w2vp translation procedure in GraphWld.tpu}
    wx, wy   : real;
    vpx, vpy : integer;

procedure AdjustParams;   {for use in rotation calcs}
begin
    SinYawRad := Sin(YawRad); CosYawRad := Cos(YawRad);
    SinRollRad := Sin(RollRad); CosRollRad := Cos(RollRad);
    SinPitchRad := Sin(PitchRad); CosPitchRad := Cos(PitchRad)
end;

procedure CalcVandDArrays;
begin
    For i := 0 to 5 do
      begin
        x  := WObj[i].x; y := WObj[i].y; z := WObj[i].z;
        x  := (-1)*x;
        xa := CosYawRad*x - SinYawRad*z;
        za := SinYawRad*x + CosYawRad*z;
        x  := CosRollRad*xa + SinRollRad*y;
        ya := CosRollRad*y - SinRollRad*xa;
        z  := CosPitchRad*za - SinPitchRad*ya;
        y  := SinPitchRad*za + CosPitchRad*ya;
        x  := x + TransX; y := y + TransY; z := z + TransZ;
        sx := Dist*x/z; sy := Dist*y/z;
        VObj[i].x := x; VObj[i].y := y; VObj[i].z := z;
        DObj[i].sx := sx; DObj[i].sy := sy
      end
end;

procedure ConstructModel;

begin
    CalcVandDArrays;
    SetColor(MaxColor);
    SetLineStyle(0,0,1);

    {Surface 0 }
    x1 := DObj[0].sx; y1 := DObj[0].sy; x2 := DObj[1].sx; y2 := DObj[1].sy;
    x3 := DObj[2].sx; y3 := DObj[2].sy; x4 := DObj[3].sx; y4 := DObj[3].sy;
    WLine(x1,y1,x2,y2);  {In GraphWld: translates params, calls Line}
    WLine(x2,y2,x3,y3);
    WLine(x3,y3,x4,y4);
    WLine(x4,y4,x1,y1);

    {Surface 1}
    x1 := DObj[1].sx; y1 := DObj[1].sy; x2 := DObj[4].sx; y2 := DObj[4].sy;
    x3 := DObj[5].sx; y3 := DObj[5].sy; x4 := DObj[2].sx; y4 := DObj[2].sy;
    WLine(x1,y1,x2,y2);  {In GraphWld: translates params, calls Line}
    WLine(x2,y2,x3,y3);
    WLine(x3,y3,x4,y4);  {Note: don't REDRAW a line to close surface}
end; {ConstructModel}

procedure WritePressKey;
begin
    SetTextstyle(DefaultFont,HorizDir,1);
    wx := 200; wy := 250;
    w2vp(wx,wy,vpx,vpy);                    {uses stand-alone translator}
    OutTextXY(vpx,vpy,'press any key...')
end;

function Deg2Rad(Degs : Real) : Real;
begin
    Deg2Rad := Degs * 0.01745327778
end;

procedure InitWorld;  {also inits graphics, program}
begin
    gdriver := Detect;
    InitGraph(gdriver, gmode,'a:\');
```


PRODUCED 2008 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
  ecode := GraphResult;
  if ecode <> 0 then
    begin
      writeln('Halted on graphics error: ', GraphErrorMsg(ecode));
        Halt(2)
      end;
    SetGraphMode(GetGraphMode);
    MaxColor := GetMaxColor;
    MaxX := GetMaxX;
    MaxY := GetMaxY;
    SetViewPort(0,0,MaxX,MaxY,ClipOn);
    {See Note in header about 'flagling' your world!}
    CreateWorld(-399.0,-299.0,400.0,300.0); {after setting viewport}
  (* CreateWorld(-399.0,300.0,400.0,-299.0); {"flipped" world}  *)

    {---Initialize DataBase---}
    WObj[0].x := 30; WObj[0].y := -30; WObj[0].z := 0;
    WObj[1].x := 30; WObj[1].y := 30; WObj[1].z := 0;
    WObj[2].x := -30; WObj[2].y := 30; WObj[2].z := 0;
    WObj[3].x := -30; WObj[3].y := -30; WObj[3].z := 0;
    WObj[4].x := 30; WObj[4].y := 30; WObj[4].z := -60;
    WObj[5].x := -30; WObj[5].y := 30; WObj[5].z := -60;

    {---assign drawing variables---}
    Dist := 1200;    {distance to picture plane}
    {YawRad := Deg2Rad(0); RollRad := Deg2Rad(0); PitchRad := Deg2Rad(0);}
    TransX := 0; TransY := 0; TransZ := 350   {Obj beyond picture plane}
  end; {initWorld}

  procedure GetParams;
  begin
    RestoreCrtMode;
    writeln('GraphWld.tpu Demo - Copyright (C) 1989 by Gene Fowler');
    writeln;
    writeln('Only the 3 rotation params to be set, not the plane and');
    writeln('object distances, translations, or placement of the two face');
    writeln('semi-cube. This side is centered on x0,y0 and all four points');
    writeln('have z = 0 - so a 90 degree yaw with 0-roll,0-pitch shows a');
    writeln('straight line. Enter all three 0s when finagling world...to');
    writeln('have an expected square for test measuring. Aspect is in world.');
    writeln;
    write('Yaw angle in degrees (0-360):  ');
    readln(YawRad);
    YawRad := Deg2Rad(YawRad);
    writeln;
    write('Roll angle in degrees (0-360):  ');
    readln(RollRad);
    RollRad := Deg2Rad(RollRad);
    writeln;
    write('Pitch angle in degrees (0-360):  ');
    readln(PitchRad);
    PitchRad := Deg2Rad(PitchRad);
    SetGraphMode(GetGraphMode)
  end;

  begin  {main}
    DirectVideo := False;
    InitWorld;
    repeat
      {set rotations, draw, and view}
      GetParams;
      AdjustParams;
      ConstructModel;
      WritePressKey;
      repeat until keypressed;
      GoAgain := ReadKey; {clear key}
    {repeat or quit choice}
      RestoreCRTMode;
      write('Repeat or quit (r/q)? ');
      repeat until keypressed;
      GoAgain := ReadKey;
      SetGraphMode(GetGraphMode);
    until (GoAgain = 'q') or (GoAgain = 'Q');
    CloseGraph
  end.
```


Total control with *LMI FORTH*™

For Programming Professionals:
an expanding family of compatible, high-performance compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers
for MS-DOS, OS/2, and the 80386

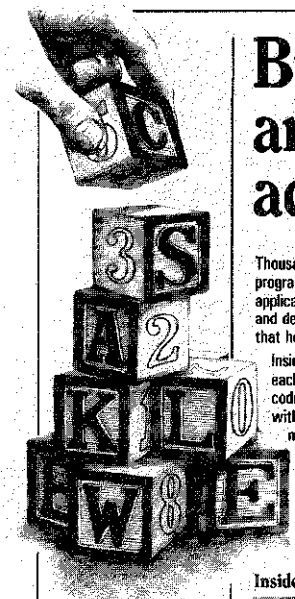
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics, floating point, native code generation

For Applications: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11,34010, V25, RTX-2000
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone Credit Card Orders to: (213) 306-7412
FAX: (213) 301-0761

Number **63** on the Reader Service Card



Build with an Inside! advantage...

Thousands of C, BASIC, FORTRAN and Modula-2 programmers are already using Inside! to improve application performance. You have the best compiler and debugger, now add the performance analysis tool that helps create world class applications!

Inside! analyzes your program in real-time, timing each instance of a function, source line or arbitrary code fragment, all with microsecond accuracy and without source code modification. Utilize the following modes to know where to focus your optimization efforts for maximum reward.

- Function/procedure timing
- Call target timing
- Source Line timing
- Arbitrary event timing
- Non-executed functions or source lines
- MS-DOS function timing
- Overlay and assembly language support

Inside! is available for popular compilers . . .

Call or write today for a free brochure and the latest list of Inside! products.

To Place Orders
(800)537-5043
Product Support
(508)478-0499
Visa/Mastercard/C.O.D. Accepted

$125.00 EACH
Satisfaction guaranteed

Paradigm Systems
P.O. Box 152 Milford, Massachusetts 01757
Inside! is a trademark of Paradigm Systems.

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Numeric Extensions, Zortech C++, and ANSI C

by

Rex Jaeschke

© *Rex Jaeschke, 1989. All rights reserved.*

In this installment, I'll introduce you to a significant new C standards activity I recently launched, and I'll tell you how Zortech's C++ compiler performed in my ANSI conformance tests.

Numerical C Extensions Group

For some time, I have had an interest in standardizing extensions outside of X3J11. Thus far, my role has been to convene an extensions meeting one evening during each X3J11 meeting week. And while each extensions meeting was well attended, only the two or three regular presenters really participated.

In September 1988, we at X3J11 decided we had done the best job we could and voted to forward the standard to X3 for final approval; although that approval has not yet been granted, all indications are that it *will* be without further committee deliberations. As a result, the X3J11 meetings will be held less regularly and for shorter periods. That, coupled with many of the regular X3J11 attendees having little or no interest in the numerical marketplace at this time, makes X3J11 an inappropriate forum for efforts to standardize extensions, at least in the numerical direction.

Certainly, any effort to standardize extensions must be done with X3J11's stated goal and future directions in mind. One of the principal reasons complex and the like are not in the first version of the standard is the lack of prior art and, in some cases, the lack of *consistent* prior art. Only when an extensions group can form consensus within its own ranks can it reasonably expect a formal standards body to consider adopting the extensions.

As a result of my involvement in X3J11, my writing and teaching interests, and my starting *The Journal of C Language Translation* (a quarterly publication aimed at C language implementers), I have formed an extensions group. The broad purpose of this "Numerical C Extensions Group" (NCEG) will be to standardize extensions to the preprocessor, language, or library (as appropriate) to sell to the numerical community and satisfy the increasing interest in C being shown by that community.

Initially, NCEG should

- Identify existing hardware and software art.
- Identify existing and pending formal and de facto standards and establish a liaison with them. (X/OPEN, OSF, POSIX, X3J11, GKS, and ISO—and maybe even FORTRAN 8X—immediately come to mind.)
- Identify the technical issues and assign them to the appropriate priority categories:
  - Must have now or as soon as possible.
  - Can live without for the time being but eventually will need or would like.
  - Wishful thinking: "Gee, wouldn't that be nice stuff?"
- Recognize, accept, and rationally deal with the problems created when competing vendors and conflicting commercial interests arise.
- Identify, as soon as possible, those areas that such a group cannot, or chooses not to, standardize.

NCEG Meeting Report

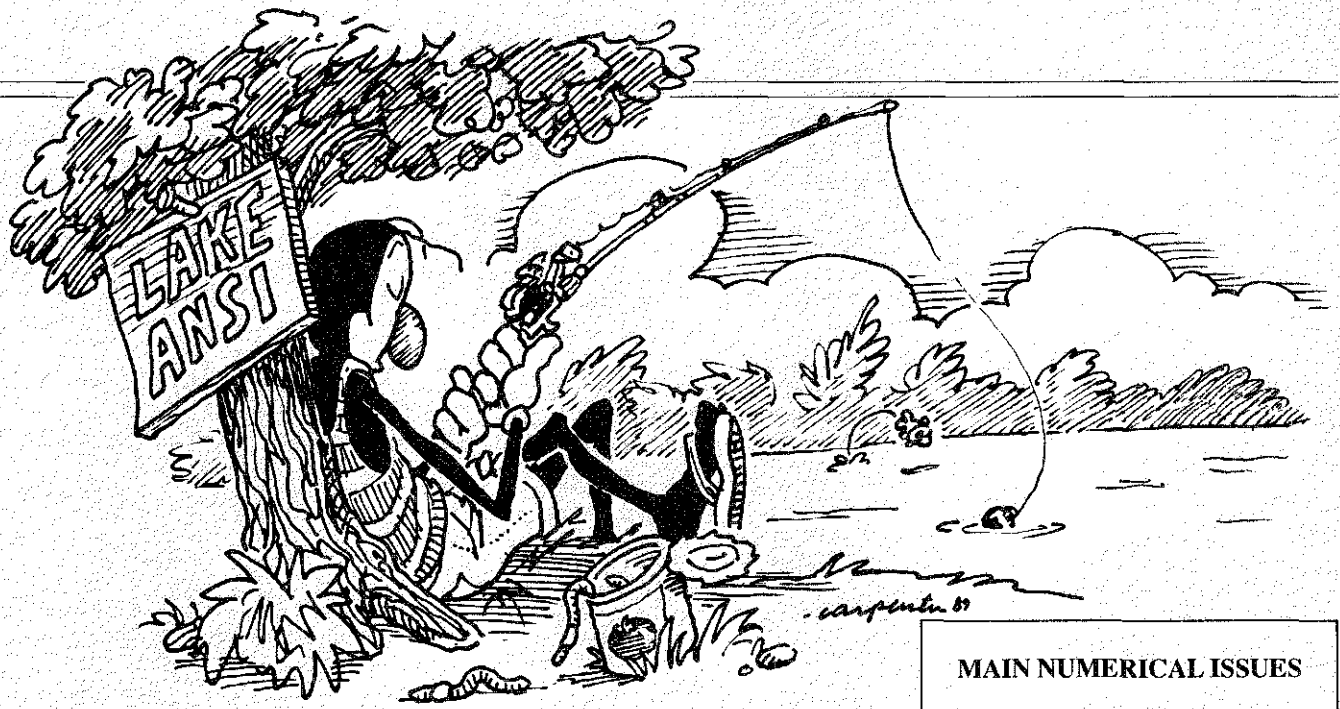
[The following report is reprinted (with permission) from Volume 1, Number 1 (June 1989) of The Journal of C Language Translation, © *1989 Rex Jaeschke.]*

Introduction

When I conjured up the idea for an ad hoc group to define numerical extensions to C earlier this year, I had no idea what the reaction would be. The evidence is now clear that this endeavor is seen as being very worthwhile: Not only have more than 90 people asked to be added to the contact database, but 30 of them attended the one-and-a-half-day meeting at Cray Research on May 10 and 11.

The backgrounds of the attendees were diverse. The supercomputing industry was represented by Cray, Convex, Supercomputer Systems, and Thinking Machines. The IEEE community was well represented by Hough (from Sun), Cody (from Argonne Labs), and Thomas (from Apple). Other organizations represented included Unisys, Microsoft, Digital Equipment Corporation, Hewlett-Packard, CDC, IBM, Solborne, Farance, the University of Minnesota, Intermetrics, and Information and Graphics Systems. The digital-signal-processing industry was represented by Analog Devices, and LLNL, Army BRL, and Polaroid Corporation represented the user com-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



munity. Dennis Ritchie, from AT&T, also participated.

There was no real sentiment for deliberately going against the direction established by ANSI C; in fact, quite the contrary. However, it was recognized that some of ANSI C's constraints may impede our activities and result in possible conflicts. The whole issue of **errno** and formatted I/O of NaNs and infinity are examples.

The Issues

The main purpose of the meeting was to identify the principal technical issues. The group then voted on each topic, indicating high or medium (or no) priority. The high-priority votes were given twice as much weight as the medium-priority votes, and the list of priorities shown in the box resulted.

Another topic, arrays as first-class objects, had a high priority (21) but after considerable debate was dropped from the list since it was agreed that its addition would likely cause great confusion for C programmers.

Formation of Subgroups

The bulk of the agenda time was then given to the top ten topics, each getting 20 to 30 minutes. For each of these topics, attendees volunteered to be primary and alternate coordinators. (The minutes of the first meeting identify these people; in the interim, contact me for details.)

The intent is for the real technical work to go on between meetings and be coordinated by the leaders of each subgroup, and then, at the following meeting, to have each subgroup present the results of its work and make formal proposals as appropriate. This way, the committee can focus on the final, distilled issues rather than having everyone get involved at all levels. It will also significantly reduce the amount of paper in the mailings.

If you wish to participate in any of these subgroups, contact the leaders and identify yourself, your concerns, and how you can help.

Mailings and Submissions

Most of the people interested in NCEG appear to have e-mail addresses, which should make coordinating various viewpoints and proposals much easier for the subgroups. However, all formal distributions will be by paper mail. Since meetings are to be held once every six months, there will be two mailings between meetings. The first will occur within four to six weeks after a meeting and will contain minutes, new papers, and other appropriate correspondence. The second will occur about four to six weeks prior to the following meeting. The cut-off date for formal submissions for the September meeting was August 11.

Forward all correspondence to me (either by mail or via uunet!aussie!rex; note that I do not have a **troff** formatter) and I will assign it a document number. However, do so only if your paper is concerned with issues not being handled by a subgroup; forward subgroup-issues papers to the appropriate subgroup coordinators so that they can include them in their submissions to me. That way, we can avoid excessive duplication of points and allow the short meeting time to be used more effectively—the more formal documents we have, the slower it will go.

Tom MacDonald at Cray Research has agreed to do the mailings, at least for the interim. Frank Farance, of Farance, Inc., has volunteered to be the redactor of the group's working document. Thanks to Tom and Frank. Thanks also to Randy Meyers from

MAIN NUMERICAL ISSUES

Topic	Priority
Aliasing	29
Vectorization	27
Complex	27
Variably dim arrays	25
IEEE issues	24
Exceptions/**errno**	24
Float/long double library	23
Parallelization	22
ANSI <math.h>	21
Array syntax	19
Extra math functions	17
Aggregate initializers	15
Interlanguage issues	15
Wide accumulators	10
Math function precision	9
Non-zero-based arrays	8
Numerical representation	6
New data types	4
New operators	4
Function overloading	4

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DEC, who acted as meeting secretary, and to Cray, for being meeting host.

Formal Affiliation

There was consensus that we become affiliated with a recognized standards organization. The final proposal was that we become a working group within X3J11. If we follow that route, it will result in our publishing a "Technical Report," a nonbinding report on our findings and recommendations. Getting our extensions adopted as a standard is also possible, in the long term. At this stage, I plan to ask for agenda time at the next X3J11 meeting to discuss admitting us as a work group.

In the interest of economy, the next two meetings are scheduled in the same location and week as those of ANSI C's X3J11. These NCEG meeting dates are September 19 and 20, 1989 (Salt Lake City), and March 7 and 8, 1990 (New York City).

Zortech C++ V1.07

Recently, I took delivery of Zortech's C/C++ compiler and put it through my ANSI conformance tests. Although both a C and a C++ compiler are included in the kit, both of them define __STDC__ to be 1 when the *-A* strict ANSI switch is selected. Since I expect more Zortech customers use the C++ component than the C one, I tested only the C++ compiler.

The Compiler Options

The only option of importance here is *-A*, strict ANSI mode. Option *-J* is supported to make plain chars unsigned.

The Preprocessor

The following ANSI C attributes are not supported or have some limitation:

- Permits a macro to be redefined differently.
- Stacks different definitions for the same macro.
- Permits the five predefined macros and defined operator to be **#undef**ed and **#defined**.
- Mishandles some backslash–newline continuations.
- Doesn't diagnose extra tokens at the end of directive lines.
- Fails to diagnose an unterminated comment.
- Can't handle **#include /*...*/ <stdio.h>**.
- Should not permit floating-point constants, casts, or enum constants in **#if** arithmetic.
- Can't handle alternate form of defined operator (without parens).
- Permits **#if**, **#else**, and **#endif** directives to be in different files.

The following test gave a surprising result:

```
#define DEF2 #define
DEF2 M2 100
```

A macro named M2 was actually created with the value 100. This is not permitted by ANSI, since all directives must exist as preprocessing tokens in the original source file—you cannot "construct" a directive name during preprocessing.

The **sizeof** operator is permitted in **#if** arithmetic, although this is not required by ANSI C.

The preprocessor passed many more tests than it failed and, in general, is in pretty good shape.

The C Language

The following ANSI features are not supported or have problems:

- No auto-aggregate initializers are permitted.
- There is no f (or F) float constant suffix.
- There is no l (or L) long double constant suffix.
- Long float is accepted but should not be.
- There is no trigraph support.

All of these are very minor issues.

The Run-Time Library

Only one header is missing, **locale.h**. However, the following headers were

Turn your favorite C compiler into a powerful database manager with the
C/Database Toolchest
The C/Database Toolchest™ adds sophisticated file management functions to your Power C™, Turbo C®, QuickC®, or Microsoft® C compiler. With the C/Database Toolchest™, your data requires much less disk space than with programs like dBASE®, and you can access your data much faster. Of course the full power of C provides you with an unlimited amount of programming flexibility.
The C/Database Toolchest™ includes three major components:
1) An advanced B+tree library gives you instant access to your data.
2) A high-level ISAM library provides you with an easy-to-use C interface, and
3) A complete database manager (with C source code included) shows you how to create impressive applications.
You also receive a comprehensive 350 page manual and a utility for converting dBASE® files.
The C/Database Toolchest™ supports features that you'd expect to find only in products costing ten times as much. Advanced features include variable length records, variable length keys, multiple keys per index, and multiple indexes stored in a single file. Your data files can contain an unlimited number of records, and each record can be as large as 32K bytes in length.
About the only thing that the C/Database Toolchest™ doesn't do is cost you a lot of money. We've kept our price low so you can manage your budget as easily as your data.
Now Only $19.95!
Order Coupon
Name
Street
City
State Zip
Telephone
Paying By Money Order Check
Visa MC AX Disc.
Card#
Exp. Date
Disk Size 5¼" 3½"
Qty Product Price Subtotal
C/Database Toolchest $19.95
C/Database Library Source $19.00 B+tree & ISAM library source code
Add Shipping ($5 USA, $20 Foreign)
Texas Residents Add 8% Sales Tax
Total Amount of Your Order
Order now by calling our toll free number or mail the coupon to:
Mix Software
1132 Commerce Drive
Richardson, TX 75081
1-800-333-0330
60 Day Money Back Guarantee
Not Copy Protected ■ Royalty Free
For technical support, please call:
1-214-783-6001
MIX software
C/Database Toolchest and Power C are trademarks of Mix Software. QuickC and Microsoft C are registered trademarks of Microsoft. Turbo C is a registered trademark of Borland. dBASE is a registered trademark of Ashton-Tate.
Number 266 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Debug Multi-Megabyte DOS Programs

There was a time when the only way to debug very large DOS C programs meant suffering under 640KB memory constraints. Not any longer. *Instant-C* opens a new age of debugging under DOS.

Instant-C is specifically designed to debug large DOS programs. It works in "protected mode" on all 80286 and 80386-based PCs giving you access to **16MB** of extended memory. *Instant-C*'s source-level debugging lets you debug and run programs in excess of 100,000 lines of code without worrying about DOS memory limitations or resorting to overlays. Hard to believe? It's true. No other DOS debugger can live up to *Instant-C*'s standards.

And *Instant-C*'s debugging capabilities are unsurpassed. With features such as automatic run-time checking, data monitoring, source code animation, source code backtracing, and unlimited conditional breakpoints at your disposal, debugging your large programs will be faster than ever before.

Instant-C also includes an integrated C compiler, linker, editor, and source formatter. With *Instant-C* you get a powerful integrated tool package for working with your C programs. And *Instant-C* is compatible with your Microsoft C compiler.

Save yourself hours of debugging time every single day. Try *Instant-C* and discover how to make debugging your large DOS C programs quicker than you ever believed possible. With *Instant-C*'s **30 day no risk money back guarantee** the only thing you've got to lose is bugs.

- **Debug Multi-Megabyte Programs**
- **Automatic Run-time Checking**
- **All Debugging in C**
- **Unlimited Conditional Breakpoints**
- **Compatible with your Assembly Code**
- **Source Code Backtracing**
- **Support for Full C Language**
- **Data Breakpoints**
- **Source Code Animation**
- **Compatible with Microsoft C**

Rational Systems' 30 Day No Risk Money Back Guarantee

If for any reason you decide not to keep *Instant-C*, just return it to us within 30 days of purchase and we will gladly refund your money.

Order *Instant-C* today!

***only* $795**

Rational
Systems, Inc.

220 North Main Street
Natick, MA 01760
Tel: **(508) 653-6006**
FAX: **(508) 655-2753**

Trademarks are acknowledged to Rational Systems, Inc., and Microsoft Corp.

Number **20** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

incomplete or contained extensions that were not conditionally compiled out in strict ANSI mode: **assert**, **ctype**, **float**, **limits**, **math**, **setjmp**, **stddef**, **stdio**, **stdlib**, **string**, and **time**. That leaves only three headers that passed all tests: **errno**, **signal**, and **stdarg**.

Almost all of the headers use a protective wrapper to prevent their being **#included** more than once. However, the names used for the macros controlling this are in the programmer's name space. For example, **ctype.h** uses CTYPE_H when it should use something like _CTYPE_H or __CTYPE_H.

Numerous function prototypes were missing the **const** keyword. From the dates in the header comments, it appears that some are quite old and haven't been revised for several years.

The header **assert.h** cannot be **#included** more than once with **NDEBUG** defined and not defined, as is required.

The definitions for CHAR_MIN and CHAR_MAX in **limits.h** are hard coded and, thus, will not be correct when the *-J* option is used. In fact, as CHAR_MIN is currently defined, it is wrong. Header **limits.h** also contains some macros now defined in **float.h**.

In **setjmp.h**, **longjmp** is declared to return an int when it should return void.

Header **stddef.h** is missing some macros and typedefs and declares **errno** (an interim location in a previous draft of ANSI C).

Lot's of work is needed here.

The Bottom Line

The C part of the compiler seems to be quite solid, and it provides reasonable error handling, for the most part. Also, many of the tricky dark corners of the standard have been correctly implemented. In terms of tracking the draft ANSI standard, it's in the top 25 percent of DOS C compilers (along with Microsoft's and Borland's products). In terms of ANSI conformance, it's still quite a ways off, although all of the immediately useful stuff is there. It is not clear why __STDC__ is defined as 1, since that means "ANSI conformant," which clearly is not true. Perhaps it's marketing license.

Rex Jaeschke is an independent computer consultant, author, and lecturer. He is actively involved with the ANSI and ISO C Standards committees and is the convener of the recently formed Numerical C Extensions Group (NCEG), which is defining numerical extensions to ANSI C. Rex is the editor of The Journal of C Language Translation, *a quarterly publication aimed at C language implementers. You can contact Rex at 2051 Swans Neck Way, Reston, VA 22091, 703-860-0091, or via UUCP: uunet!aussie!rex.*

Free Money!

Well, almost free—
Your product can present a consistent high-quality user interface, with a minimum of effort on your part—
It's like getting paid for not working!

ScreenLib™
Complete C Source Code!
No Royalties!

Simple Screen Definition ★ Windows
Pop-up Menus ★ Context-Sensitive Help
Lots of low-level functions

Introductory Price: $69.95
Plus $5.00 shipping ($10.00 outside US)
CA residents please add sales tax

Business Computer Services
1800 S. Robertson Blvd., Suite 206
Los Angeles, CA 90035

VISA/MC orders: (213) 836-5026
Demo disk available—$10.00

Number **286** on the Reader Service Card

VIDEO TAPE

Understanding C. 3 volumes, 6 hours, covers every nook and cranny of ANSI C. Loaded with examples! Results Guaranteed! Comes with reference book. You'll learn techniques used by experienced C programmers. $59.95 post paid. (COD at No Extra)

To order COD, call (817) 496-8747
or send check or money order to:
AUTOMATION EDUCATION
2309 Royce Drive
Arlington, Texas 76016-1232

Number **207** on the Reader Service Card

Parallel Programming for "C"

INTERWORK™
A Concurrent Programming Toolkit

Interwork is a "C" program library which allows you to write your programs as a set of cooperating concurrent tasks. Very useful for simulation, real-time applications, and experimentation with parallel programming.

FEATURES

- Supports a very large number of tasks (typically more than 100) limited only by available memory. Low overhead per task results in very fast context switching.
- Provides a full set of inter-task communication (ITC) facilities, including shared memory, locks, semaphores, blocking queues, and UNIX™-style signals. Also has building blocks for constructing your own ITC facilities.
- Handles interrupts (DOS version) and integrates them into task scheduling. Supply your own interrupt handlers or block tasks on interrupts.
- Lets you trace task switches and inter-task communication.
- Comes with complete documentation including a user's manual and reference manual of commands.

Interwork is available for the following systems:

Hardware	*Operating System*	*Price*
IBM PC, XT, AT	PC-DOS 2.0 or later	$129
IBM PC AT	XENIX™	$159
DEC VAX™, SUN 3, 4	UNIX 4.2BSD	$249

PC-DOS version is compatible with DeSmet, Lattice, and Microsoft C compilers.

Please specify hardware and operating system when ordering. Shipping and handling included; COD orders add $2.50. Send check or money order to:

Block Island Technologies
Innovative Computer Software
15455 NW Greenbrier Parkway, Suite 210, Beaverton, Oregon 97006
(503) 690-7181 FAX (503) 645-7732

Trademarks: Interwork, Block Island Technologies; UNIX, AT&T Bell Laboratories, Inc.; XENIX, Microsoft, Inc.; VAX, Digital Equipment Corporation

Number **244** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386 Development Environments and Tools

Soft-ICE 2.0 and MagicCVW
Nu-Mega Technologies
P.O. Box 7607
Nashua, NH 03060-7607
603-888-2386

Nu-Mega Technologies has recently released version 2.0 of Soft-ICE, their powerful DOS debugger that uses the 80386 to provide hardware-debugging capabilities. On an 80386 machine, Soft-ICE can add some very sophisticated features to your favorite DOS debugger. These features are normally found only in hardware-assisted debuggers and In-Circuit Emulators (ICEs). These features include real-time breakpoints on memory reads/writes, port reads/writes, memory ranges, and interrupts. Soft-ICE also adds the ability to break out to the debugger with a keystroke even if interrupts are disabled and your application has taken over the keyboard interrupt. Note that Soft-ICE will work with real-mode programs only; it will not work with programs that use 80286 or 80386 protected-mode instructions.

Version 2.0 has added back-trace history ranges, EMS 4.0 support, extended-memory support, symbolic and source-level debugging, and an enhanced user interface. The back-tracing features provide a memory history of what your code executed, along with a trace simulation through your source. Version 2.0 now includes a full EMS 4.0 driver, plus powerful EMS debugging features. Soft-ICE now allows you to look at and modify all of the available extended memory, which is a very useful tool for debugging VDISK, CACHE, and other programs that make use of extended memory. The new user interface with version 2.0 now has separate Code, Data, and Register windows, programmable function keys, and "point-and-shoot" breakpoints. The new source-level features are somewhat limited, but they do provide the ability to debug device drivers, interrupt handlers, TSRs, and ROMable code more easily.

The actual debugger code is completed isolated from your executing real-mode code, preventing an errant program from modifying or destroying Soft-ICE, even if DOS is clobbered. This new version can be configured to back-fill base memory past 640K and use none of the lower 640K of lower memory for the actual debugger code. Soft-ICE can also be used as a stand-alone debugger with a soft-boot capability for non-DOS systems. Soft-ICE can be loaded as a device driver or invoked from the command line as a TSR. When loaded as a TSR, it may be later unloaded from memory, but it must be loaded as a device driver when installed with other drivers (VDISK.SYS, RAMDRIVE.SYS, HIMEM.SYS, EMM 4.0, etc.) that use extended memory using ROM BIOS calls and when using the new history, symbolic, and source-level debugging features. A utility program converts the link-map file into a symbol file with line numbers and public symbols for stand-alone debugging.

The real strength of this product is found in using it with your favorite symbolic debugger. Nu-Mega also offers a companion product called MagicCV that allows running Microsoft Codeview in only about 8K of DOS memory. But the real triumph for debugging is a brand-new product, MagicCVW, that runs Codeview for Windows in only 8K of DOS memory. These two products together (Soft-ICE 2.0 and MagicCVW) provide possibly the best of all worlds if you're developing Windows-based applications. Soft-ICE comes with a solid, 270-page manual with a decent index and a quick-reference card. MagicCVW comes with a short, 32-page companion manual. Soft-ICE is available for $386, and MagicCVW costs $199; combined, the two products are available for $499.

OS/286 and OS/386 Developers' Kits
Eclipse Computer Solutions
One Intercontinental Way
Peabody, MA 01960
508-535-7510

Eclipse, formerly A. I. Architects, has just released Version 2.0 of the OS/286 and OS/386 Developers' Kits. These products provide DOS extenders for both 80286- and 80386-based DOS machines. As I noted in previous articles, the Eclipse OS/386 is the DOS extender that is closest in implementation to native DOS. Version 2.0 is a major upgrade to both the 286 and 386 versions of their DOS-extender family. New features in Version 2.0 include an automatic configuration program, support for Virtual Control Program Interface (VCPI), reduced DOS-memory usage, enhanced memory management, easier interrupt support, virtual 8086 mode for DOS programs, greater machine compatibility, improved real- and protected-mode communication, and new language support.

The kernel is now installed as a TSR, which may be removed from memory when not needed. This speeds the loading of protected-mode programs during both development and use. The OS/X86 DOS extenders now adhere to the VCPI specifications, allowing them to run under Quarterdeck's DESQview 386, which provides multitasking capabilities. The VCPI adherence also provides for compatibility with other conforming DOS extender applications. The resident OS/X86 kernel can now be configured to use only about 40K of the lower 640K DOS memory space, freeing more memory for other real-mode applications and TSRs. New memory-management features include multiple heaps, automatic compaction, and control over where the protected-mode component of the kernel is loaded—below or above the 1-megabyte boundary. New memory-management service functions allow access to page tables, selection of low or high memory heaps, and control over compaction. Interrupt handlers may be easily chained to real-mode handlers or shared between parent and child tasks. The virtual 8086 mode is supported, allowing up to 900K of memory for DOS applications (paging is used). This new release is compatible with the widest array of 286-clone machines. A supplied setup program automatically tunes protected-mode applications for the optimum performance on 286 and 386 machines. New language support has been added for Borland Turbo C and Turbo Pascal, and support will soon be available for WATCOM C and the new Lattice C compilers. Microsoft C and FORTRAN, MetaWare High C, and older versions of Lattice C continue to be supported. Lahey FORTRAN uses OS/X86 for their extended-memory FORTRAN compilers.

The OS/X86 DOS extenders support a powerful Remote Procedure Call (RPC) mechanism that facilitates communicating with real-mode code, such as graphics (samples for Halo and GSS are provided), communications, and so on. This mechanism has been enhanced, allowing RPCs to be written in C (Microsoft C is currently supported) rather than assembly language. Both DOS extenders come with a powerful debugger CP that provides a sophisticated EMACS-style editing environment for the command line along with loadable macros for symbol substitution. A utility converts a link map or object file into a symbol file suitable for use with CP, providing symbolic debugging.

Continued on page 91

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

T-SHIRTS

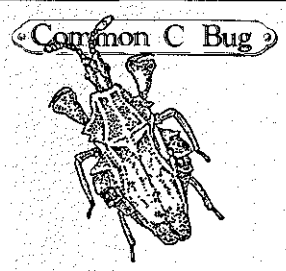
Multi-color imprints on the highest quality 50/50 cotton T-shirts.

$12.00 each – includes shipping & handling

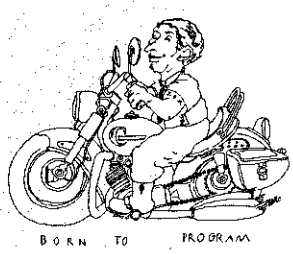
SIZES S, M, L, XL

PJ ISSUE SPECIAL!

$5.00 OFF REGULAR PRICE
42% savings on 6 individual issue disks!



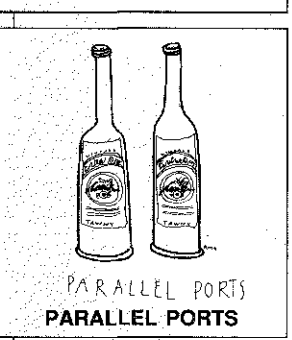
COMMON C BUG



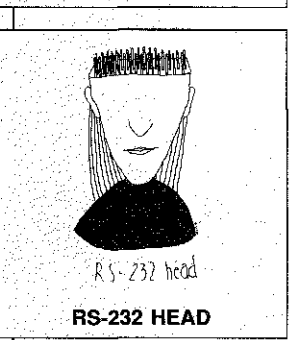
BORN TO PROGRAM

MORE CODE FOR LESS COST!

Save **$5.00** off the regular price of a ***PJ Code DiskScription.*** Get all of the program listings (plus additional .EXE, .OBJ, and benchmark source files) published in the next 6 issues of ***PJ.***
$34.95 for U.S., $44.95 in Canada or Mexico,
$49.95 all other foreign.
(Offer expires October 31, 1989. All orders must be prepaid.)



PARALLEL PORTS



RS-232 HEAD

BUGS RULE 4 color imprint on a white shirt

BACK ISSUES

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 5.5: Intel's 80386 vs. OS/2; interview with Metagraphics President Jack Davis; Consulting—making computer tools pay for themselves; ASM—the invisible CapsLock; TSRs using expanded memory with Turbo Pascal; note on optimizing Turbo Pascal; proposed ANSI C Language Standard; printer output from Turbo C programs; match-up of Turbo Basic 1.0 and QuickBASIC 3.0.

Vol. 5.6: Multics, OS/2, paging, and segmentation; The Usenix conference and other news; how to get rich in consulting; Shell in Marshal Pascal; Turbo Pascal 4.0 preview; handling fixed-length character arrays in C; the proposed ANSI C language standard.

Vol. 6.1: Consulting is an attitude; write mode 3 of the VGA; the LIM 4.0 expanded memory specification; environment variables in Turbo Pascal 4.0; a timer program in C; porting mainframe software to the PC—FORTRAN compilers under DOS and the Definicon 32-bit coprocessing engines.

Vol. 6.2: Hal Hardenbergh on RISCs, CISCs, and Fabs: Pink Noise; yet another VGA write mode; inside Turbo Pascal 4.0; the proposed ANSI C language standard; converting a Turbo C program to a TSR; 80286 and 80386 programming; 80386 software development; *The Norton Guide for BIOS/DOS Assembly.*

Vol. 6.3: Hal Hardenbergh on memory trends; Pink Noise; getting your consulting tools in order; Gordon Letwin on OS/2 memory management; virtualizing the 386; Pascal—a CASE of the jumps; Turbo Professional 4.0; BASIC—CALL, GOSUB, and multiline functions.

Vol. 6.4: Job shopping; windowing systems overview; reading VGA memory; X Window—a view from DOS; a digital clock in C for MS Windows; windowing in TopView, DESQview, and MS Windows; MS C 5.1 and the ANSI standard; a mouse unit for Turbo Pascal 4.0.

Vol. 6.5: Hal Hardenbergh on computers in the fast lane; Consulting—no-win jobs and too much work; VGA text paging; using dynamic arrays and records in Turbo Pascal 4.0; an OS/2 Presentation Manager application template; WATCOM C 6.0 and the ANSI standard; object-oriented OS/2 in C++; Ada and software engineering; REXX on the PC.

Vol. 6.6: PC DOS 4.0; OS/2 vs. UNIX; VGA screen saving and color selection; MASM quirks; addressing extra memory for DOS; DOS device drivers debugging and information utility; proposed ANSI C language standard; Ray Duncan on Forth; Ada data types and exceptions; PC floating-point hardware and software.

Vol. 7.1: GRAPHICS: High-resolution VGA color modes; graphics processors; modifying type constants stored in Turbo Pascal 4.0 and 5.0 compiled EXE code; comparing Pascal overlay managers; JPI's TopSpeed Modula-2; faster QuickBASIC code; Microsoft's CodeView debugger; Lattice C V3.3; smart terminal emulations in Windows; the COBOL-85 ANSI standard; Pink Noise—industry news and reviews.

Vol. 7.2: OPERATING SYSTEMS: undocumented DOS features; fast 80386 block moves under DOS; OS/2 dynamic link libraries; OS/2 multitasking; debugging utility for PM and Windows; EGA and VGA animation; disk formatting in Pascal; how to become a reseller; reviews of windowing and graphics tools, disassemblers, and *The 80386 Book.*

Vol. 7.3: DATABASE JUNGLE: neural nets; Michael Abrash's "On Graphics"; embedded SQL for a relational database; Btrieve; converting database files using filters; the proposed ANSI C language standard; a substitute for the missing TYPE command in Windows; user interface tools for DOS; reviews of dBASE C libraries, software tools, FORTRAN tools, conferences, exhibits, and books.

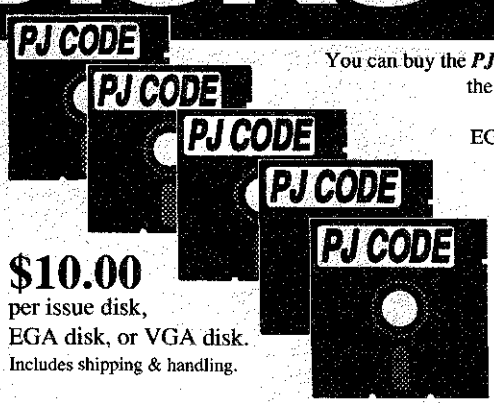
Vol. 7.4: COMMUNICATIONS: "Pink Noise," by Frank Greco; Michael Abrash's "On Graphics"; a simple terminal program in PM; communications and protocols; better coding conventions for assembly language; implementing the Boyer–Moore search algorithm in assembly language; database normalization theory and practice; Turbo Pascal 5.5 development tools; Lint utilities for the PC; a millisecond timer for BASIC; exit codes for QuickBASIC; reviews of RamNet, DOS/16M, PCX *Technical Reference Manual* and Programmer's Toolkit, GRASP 3.1, AXOS, and *Microsoft Mouse Programmer's Reference.*

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

All mail orders must be accompanied by check or Money Order – PREPAID ONLY.

CODE DISKS

Use the order form on the card opposite.
Order by phone with your VISA/MC.
CALL TOLL FREE
800/234-0386 ext 53



$10.00
per issue disk,
EGA disk, or VGA disk.
Includes shipping & handling.

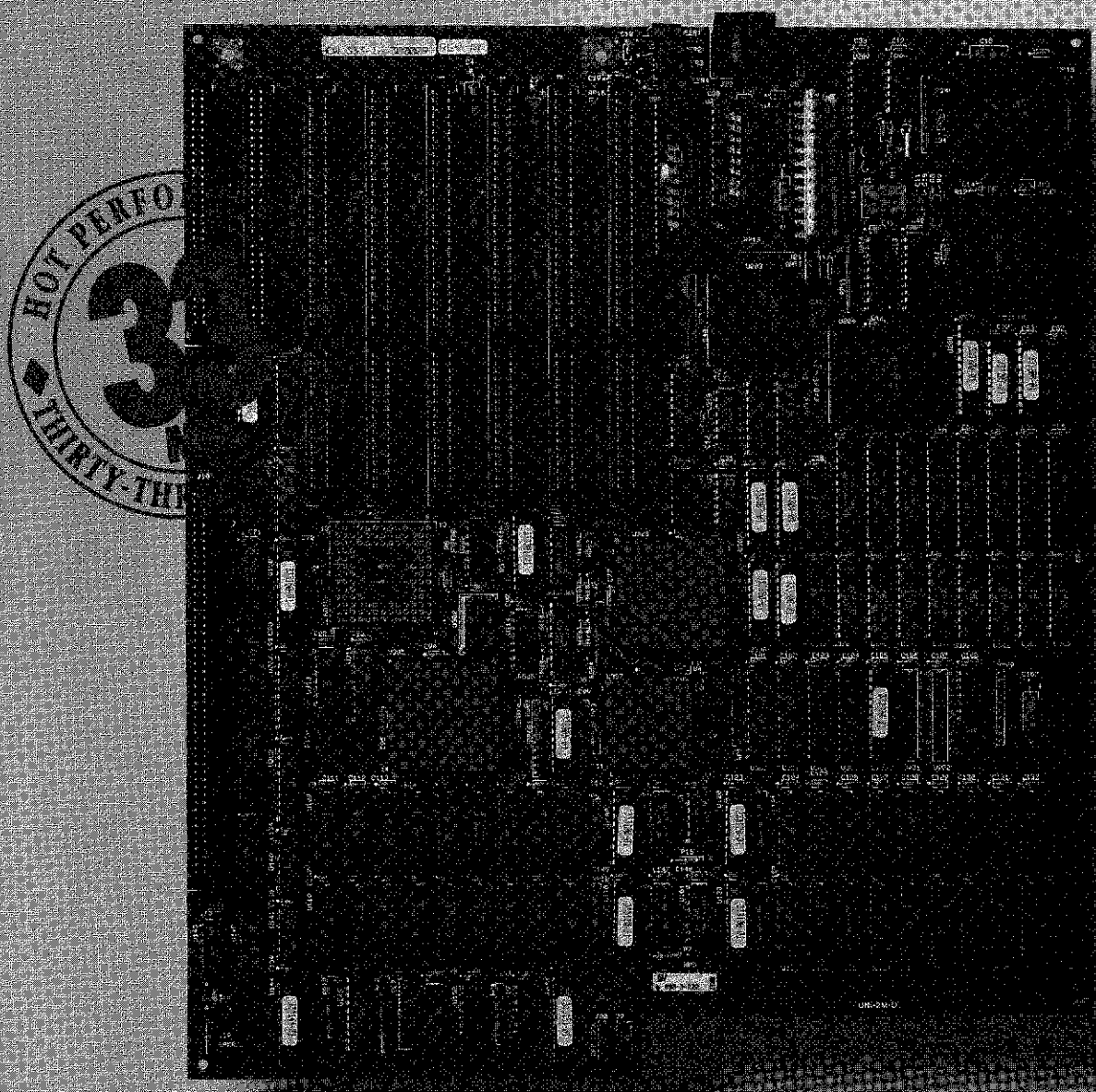
You can buy the *PJ* code listings from volume 5.5 to the current issue. Code disks for the Michael Abrash "On Graphics" EGA/VGA series are also available.

Diskettes are available for the following issues:

5.5	6.4	7.3
5.6	6.5	7.4
6.1	6.6	7.5
6.2	7.1	EGA
6.3	7.2	VGA

360K MS-DOS disks standard; other formats available.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



IT Hz SO GOOD!

If your computer feels slow, we know where it hertz. For a fast cure, get our new 386 MotherBoard/33MHz. It will accelerate your PC/AT to speeds equal to the IBM Model 70. And faster. We've built in 4 Megabytes of high speed RAM, 64K of RAM cache, and a 387 math coprocessor socket for blistering performance!

Network Savvy. With the 386 MotherBoard/33MHz, your computer becomes a file server or workstation that makes Novell networks *scream*. Beyond speed, you'll enjoy compatibility with Token Ring, Ethernet, Arcnet, and most other network cards.

The UNIX Engine. Great for VARS, Systems Integrators and UNIX OEMs, the Hauppauge 386 MotherBoard/33MHz runs SCO Xenix, Interactive 386/ix and AT&T's UNIX System V. With its PC/AT compatible I/O system, our 33MHz board accommodates the latest in disk control, graphics, and network I/O cards — just to name a few.

CAD Capability. Ever find yourself growling "faster, faster!" at your AutoCad and other CAD programs? We've got good news. The 386 MotherBoard/33MHz boosts your math and graphics applications, and supports the high speed 387-33 and Weitek Abacus math coprocessors.

Technical Features The 386 MotherBoard/33MHz includes: ▪ 4 Megabytes of high speed 32-bit memory, expandable to 64 Megabytes ▪ 64K of 20 nsec cache memory ▪ Six 16-bit expansion slots, one 8-bit and one 8-bit/32-bit slot ▪ PC/AT compatible I/O system for support of OS/2 and UNIX.

386 MotherBoard/16MHz for the PC, XT, AT and Compaq portable	$1495
386 MotherBoard/20 and 25MHz for the PC/AT	from $1895
386 MotherBoard/33MHz with 4 MBytes of RAM	$4495
32-bit RAM Boards	from $ 295

Prices subject to change

Why endure a painfully slow computer? With Hauppauge's 33MHz MotherBoard, your computer Hz so good! For more information and fast delivery, call: **1 (800) 443-6284**. In New York: (516) 434-1600. Available through Egghead Direct 100 programs, and from MTI Systems (1-800-645-6530.) Also available on GSA contract #GS00K-89AGS0358.

Hauppauge Computer Works Inc.
175 Commerce Drive,
Hauppauge, New York 11788

Hauppauge!

Number 122 on the Reader Service Card.

Trademarks: IBM PC, XT, AT, PS2 and OS/2 IBM, Intel, 386, Intel Corp., Windows - 386, Microsoft Corp. DESQview, Quarterdeck

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED.

Six easy ways to boost your BASIC

PROBAS™ Basic Programming Library

So who cares that BYTE magazine calls PROBAS a "Supercharger for QuickBASIC" or that PC Tech Journal says that PROBAS is a "high-quality, high-quantity package"? Who buys a product just because Jerry Pournelle said "Anyone doing serious QuickBASIC programming would do well to get [PROBAS]"? And who cares that Wayne Hammerly calls PROBAS "The greatest thing since sliced bread"?

Who?--Only those who want to write better, faster, slicker programs and save hundreds of programming hours in the process. With all of that hoopla out of the way, we are formally announcing the momentous release of PROBAS Version 3.1, now with over 400 assembly routines to make BASIC programs faster and more powerful than you ever dreamed with features like:

- A 1,000-page two-volume manual
- Full mouse support
- Extended and EMS memory support
- Full-featured windowing
- Moveable, resizable windows
- Screen snapshots (text & graphics)
- Virtual screens in memory
- Lightning-fast file I/O
- Critical error handling
- String, array, and pointer sorts
- Search directories and archives

Create dazzling screens in text, CGA, EGA, VGA, and Hercules graphics modes with windows that can overlay one another and be moved and resized on the fly. Store megabytes of string, data, or screen snapshots in extended or EMS memory. Draw complex text or graphic screens to memory and snap them on in an eyeblink. The PROBAS file I/O routines allow you to read or write huge chunks of data at a clip, far faster than with BASIC.

PROBAS also has over 300 other essential services, including handy string, date, time, directory, and array manipulation routines; string, screen, and data compression routines; valuable equipment and input routines; and faster replacements for many BASIC commands.

Whether you are a professional or a novice, PROBAS will boost your BASIC in ways you never thought possible. PROBAS allows the professional to write faster, tighter code in much less time and allows novices to quickly and easily write professional-quality programs that would be impossible with BASIC alone. The bottom line is PROBAS adds power and saves time. After all, how much is a few hundred hours of your time really worth?

For all DOS versions of QuickBASIC and BASCOM. ***Just $149.00!***

Number 248 on the Reader Service Card.

PROREF™ On-Line Help For PROBAS

PROREF provides on-line help for the routines in the PROBAS library. This hypertext manual links directly to the QB Advisor in QuickBASIC 4.5 so that the PROBAS reference becomes an integral part of your QuickBASIC on-line manual. Includes information and examples on PROBAS routines and helpful hints on programming in BASIC. ***Just $50.00!***

PROSCREEN™ Screen Management

PROSCREEN is a full-featured screen generator/editor that will save you more design and coding time than you ever thought possible. PROSCREEN treats screens like a word processor treats text to provide complete control over characters, colors, and placement. Design input screens with up to 130 fields, 19 predefined and 2 user-defined masks. Save screens to screen files or .OBJ files and use the tight BASIC/Assembly code that comes with PROSCREEN to access the screens. There's no kludgy code generator here! Access hundreds of input screens with less than 25k of total code. ***Just $99.00!***

PROMATH™ Mathematics Library

PROMATH is a collection of over 150 high-level routines that provide mathematical functions and operations for programmers who often work in mathematics, science, or engineering. Complex variables, real and complex matrices, real and complex trigonometric and hyperbolic functions and their inverses, solution of linear equations, integration, differential equations, Fast Fourier transforms, graphics support, and many other useful routines are provided.

For years Fortran has been the language of choice for scientific and engineering applications, but it lacks many of the useful features of QuickBASIC. PROMATH contains most of the Fortran mathematical and numeric functions and allows you to easily translate Fortran code to BASIC or write new programs in BASIC, while retaining Fortran's numerical prowess.

The PROMATH manual is over 200 pages and provides a complete description of each routine, including any algorithm and the mathematical formula the routine uses, shown in standard notation. For QuickBASIC 4 and BASCOM 6 only. ***Just $99.00!***

PROBAS™ TOOLKIT

The TOOLKIT is a collection of high-level BASIC and assembly modules that use the routines in the PROBAS library to save you even more hours of grunt work. Why spend hundreds of hours re-inventing the wheel when you can just plug in TOOLKIT modules like:

- Super-fast B-Tree indexing
- Ring, Bar, Pop-Up, Pull-Down menus
- Scroll-bar tag windows
- Dialog boxes with radio buttons
- Two mini-editors with word wrap
- BCD math routines
- Julian date & calendar routines
- Patch .EXE files
- Protected memory storage area

The TOOLKIT now supports EGA and VGA graphics modes for menus, windows, editors, calendars, and more. Complete with BASIC source code and an all-new comprehensive manual. The TOOLKIT requires the PROBAS library and helps conserve your greatest asset of all—time! ***Just $99.00!***

PROBAS™ TELECOMM TOOLKIT

The PROBAS TELECOMM TOOLKIT is a collection of high-level communications modules that you plug into your code to provide popular file transfer protocols, terminal emulations, login scripts and baud rates up to 115,200 baud. You get:

- Xmodem/Modem7/Xmodem-1k
- Ymodem (single and batch)
- CRC-16 and Checksum
- VT52, VT100, ANSI, BBS etc.
- Auto Dialer & data base
- Documented BASIC source

Why use clumsy SHELLs to complex terminal programs when you can plug just the communications routines you need into your code? Implement just the features and commands you want. Requires PROBAS. ***Just $75.00!***

Our thirty-day, money-back guarantee assures you the highest quality and our technical support staff is always ready to help.

HAMMERLY
COMPUTER SERVICES, INC.
9309 JASMINE COURT • LAUREL, MD 20707

(800) 343-7484

INT'L. ORDERS: (301) 953-2191 FAX: (301) 725-8147
BBS: (301) 953-7736

Add $5.00 per item ($6.00 Canada) for shipping per order. Europe: $35.00 for 1st item plus $5.00 for each additional item. Visa, M/C, C.O.D. (US Only) checks and approved POs accepted. Trademarks PROBAS, PROREF, PROSCREEN, PROMATH: Hammerly Computer Services, Inc. QuickBASIC, BASCOM: Microsoft Corp.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The new OS/X86 versions come with a newly revised and expanded manual with a decent index. Eclipse provides technical support and has maintained a bulletin board for some time. The run-time license fee structure of this new version has been dramatically modified, making OS/X86 much more attractive for small developers: For an installed user base up to 2,500, the run-time license fee is just $10—total, not per copy. For 2,500 to 10,000 users, the license fee is fixed at $1,500. The OS/286 Developers' Kit is specially priced at $195 through September 30; $495 thereafter. If you've been planning on trying out a DOS-extender environment for your growing application, OS/286 or OS/386 just might be the ticket.

FTN77/386
OTG Systems
P.O. Box 239
Clifford, PA 18413-0239
717-222-9100

If you haven't yet heard, there is another 80386 FORTRAN compiler on the block. OTG Systems has been marketing FTN77/386, a 386 FORTRAN compiler developed at the University of Salford, in Great Britain. FORTRAN compilers from the University of Salford have been available on minicomputers for some time. Their Prime minicomputer FORTRAN compiler has a better reputation than a similar product developed by Prime. This is their first foray into the PC arena. FTN77/386 uses a DOS extender also developed at the University of Salford. FTN77/386 supports the 80287/80387 math coprocessors required for operation. Weitek support is not currently available.

These 386 development tools (compiler, linker, debugger, etc.) form quite a complete development environment. For instance, all these utilities have an on-line help facility built in. The 386 DOS-extender run time is installed as a TSR and contains much of the FORTRAN and DOS run-time libraries. The compiler supports a load-and-go option for quick testing. Compile speeds are very fast, equal to the fastest 386 FORTRAN compilers from Lahey and SVS. The compiler provides a large number of options supporting many FORTRAN extensions and some useful debugging aids. These aids include run-time checking of array subscripts, arithmetic overflow, and argument-consistency checking. In addition, variables and array elements can be checked to be sure that they are given a value before being used. Language extensions include long names, various data types (e.g., INTEGER*2, INTEGER*4, LOGICAL*2, LOGICAL*4), a WHILE statement, and very extensive subroutines, intrinsic functions, and internal procedures.

The compiler also supports an in-line 80386 assembly language syntax of limited power. A large number of DOS and PC-system routines are already provided. The in-line assembler and large set of DOS-support routines are really mandatory, since the 386 DOS extender provided is not supported by any of the other 386 development tools from, say, Phar Lap (80386 assembler) and MetaWare (386 C compiler). Included are support routines for bit handling, storage management, character handling, screen/keyboard, time/date, file manipulation, command-line parsing, and window manipulation; graphics, mouse, in-line, and miscellaneous routines are also provided—a very rich set of primitives for a FORTRAN compiler.

Contrary to what may have been written in an earlier *Computer Language* review, the compiled FORTRAN executable code is about average in speed for 386/387 compilers. INTEGERs default to INTEGER*2, and so *Computer Language*'s results for Whetstones are significantly overrated (INTEGER*2 values truncate the loop counts in the Whetstone benchmark). Also, FTN77/386 lacks a LOGICAL*1 data type, which may pose problems when porting from VAX or other minicomputer FORTRAN code. Documentation consists of a solid manual of over 200 pages with a decent index. Documentation on DBOS (their 386 DOS extender) is somewhat limited, and I figured out quite by accident how to unload the 386 run-time TSR from memory. FTN77/386 is available for $1,095, and the run-time is available for distribution with your application, in blocks of five, for about $50 per copy. The largest limitation of this product is that the 386 DOS extender supplied is of OTG's own design and so is not supported by other 386 development tools (such as macro assemblers and C compilers).

Mini-Case Tools

CC-RIDER
Western Wares
Box C
Norwood, CO 81423
303-327-4898

Western Wares has released both DOS and OS/2 versions (1.2) of their CC-RIDER package. CC-RIDER consists of a sophisticated C source file symbol analyzer, which creates a database for your C source, along with a powerful pop-up symbol cross-reference utility usable from within your favorite editor. The symbol analyzer (CCSYM) performs a complete syntactic parse of your C source code application modules, as well as of any #include files necessary for compilation. The symbol parser is compatible with the proposed ANSI C standard and also supports extensions used by Microsoft and other C compilers, such as near, far, pascal, interrupt, and cdecl.

CCSYM reads your C source and include files as input, like a compiler, but instead of producing object code, CCSYM analyzes your C symbols. CCSYM can produce a

Unleash the BASIC Power of Hypertext!

Our latest BASIC-booster includes the first hypertext engine designed to be called from BASIC. With the **ProBas HyperHelp ToolKit** you can use the hypertext engine to put one or more manuals on-line with full hypertext search facilities. Imagine having one or more manuals on-line with all of the fantastic hypertext abilities in the QuickBASIC 4.5 compiler and then some:

- Choose single- or multi-window display
- Specify window colors and placement
- Move and resize windows with mouse or keys
- Sophisticated mouse and keyboard interface
- Up to 40 bookmarks to move between hyperlinks

In less than a dozen lines of code you can pop-up context-sensitive help at any time. Your users can then jump to related help, examples, or just browse the manual(s) in one or more windows that they can move or resize at will. The text will automatically wrap within the window to stay fully visible!

Create HyperCard Applications

Use the **HyperHelp** engine to create full-blown hypercard applications. Create sophisticated multi-window, multi-thread hypercard stacks. Mix cards and manuals for total data integration. Moving from link-to-link or card-to-card is instantaneous and the speed will amaze you.

Adding hypertext to your applications takes less than about a dozen lines of BASIC code. Converting ASCII text to hypertext is just as easy—just put delimiters around keywords, hyperlinks, and items you want to display in boldface.

Multiple Help Subsystems

The hypertext engine is just a part of the **HyperHelp ToolKit.** There is a wide selection of help subsystems with various displays, user interfaces, and memory requirements to suit almost any need. Want a small, window-oriented, keyword help system? It's in there! Need a Terminate & Stay Resident help system? It's in there! How about a lightbar indexing system that then pops-up the selected text? It's in there!

The **HyperHelp ToolKit** gets it's blinding speed by using the low-level routine in our **ProBas** Professional Basic Programming Library. **HyperHelp** requires the **ProBas** Library. See our ad on the opposite page for information on **ProBas,** shipping rates, and our thirty-day money-back guarantee.

Special Introductory Price—For a short time this powerful hypertext engine and collection of help subsystems is available for ***just $99.00!***

HAMMERLY
COMPUTER SERVICES, INC.
9309 JASMINE COURT • LAUREL, MD 20707
(800) 343-7484
INT'L. ORDERS: (301) 953-2191 FAX: (301) 725-8147
BBS: (301) 953-7738

Add $5.00 per item ($8.00 Canada) for shipping per order. Europe: $39.00 for 1st item plus $5.00 for each additional item. Visa, M/C, C.O.D. (US Only) checks and approved POs accepted. Trademarks ProBas, ProRef, ProScreen ProMath: Hammerly Computer Services, Inc. QuickBASIC, BASCOM: Microsoft Corp.

Number **293** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

complete log file of your system's modules, headers, and symbols. The log file reports statistics on usage of symbols—macros, functions, structures, unions, enums, typedefs, and so on. CCSYM generates function prototypes conforming to K&R C or the newer ANSI C prototype style.

CCSYM can also generate or update a special symbol database containing information on macros, typedefs, functions, variables, structures, enums, and so on, used by a series of C source files. Once created, this database can be used by CC-RIDER, the pop-up editor utility. Wildcards are supported on the CCSYM command line, allowing you to generate a complete database file in one invocation. A list of input files can also be read from a file.

I found the CCSYM parser to be pretty robust: It stumbled on some nonstandard C code (#ASM directives) but continued parsing the file without aborting. Include files are found using the INCLUDE environment variable. CCSYM does not gather statistics on auto (local) variables declared within a function.

CC-RIDER provides instant access to all macro, typedef, scalar, function, structure, union, and enum definitions from within your favorite editor. To look up a symbol, just move the cursor into the word of interest and activate CC-RIDER with a hot-key sequence (which defaults to Alt+Left-Shift for convenience). If a definition exists for the symbol, a larger window will appear, showing the symbol definition. At this point, you can choose to view the original source file where the definition was found and even scroll around in this file. Cut-and-paste features are available. When you're done viewing information, pressing a return key or two will restore the screen and put you back into your favorite editor.

The CCSYM utility executes reasonably fast and allows incremental updates to the symbol database. The DOS version of CC-RIDER (the pop-up TSR) requires about 150K of memory, which appears excessive, given the application. The OS/2 version uses some OS/2 monitor features and acts as a multifunction application. Both versions come with a short, 42-page manual with a decent index. I found the pop-up utility quite well implemented and very useful—a way to simulate hypertext editing within your own favorite editor. The DOS version costs $89, while the OS/2 version costs $179.

EasyFlow
HavenTree Software Limited
P.O. Box 1093-V
Thousand Island Park, NY 13692
800-267-0668
$149.95

HavenTree Software has released Version 6.0 of EasyFlow, an on-screen interactive graphics program designed to compose and modify charts and diagrams easily. While other programs may generate diagrams from your existing code, EasyFlow can be used by programmers to generate design flow charts and data flow diagrams *before* the actual coding begins. This new version has a number of enhancements that will benefit programmers.

EasyFlow is not simply another graphics program that draws boxes and connecting lines. The program has extensive charting knowledge built in and takes much of the burden from making flow charts and diagrams. Some of these features include automatically routing lines, automatically centering text within shapes, and automatically placing text blocks along lines. While the program works fine with only a keyboard, a mouse is fully supported.

While previous versions supported only shapes used for flow charts, this new version now supports data-flow diagrams. Chart shapes are organized in three separate libraries: flow charts, data-flow diagrams, and organizational charts. The shape libraries cannot be mixed. You create charts by placing shapes, lines, and text on a scrollable screen. Hercules, CGA, EGA, and some enhanced-color video cards are supported. In a 640K machine, the chart size is limited to about 300 shapes, but EasyFlow can be configured to use expanded memory to increase the maximum chart size. The flow chart shape library sports 34 different shapes, about one-third of which are available in three sizes. Dot-matrix, LaserJet, and PostScript printers are supported as output devices. Plotters from Hewlett-Packard and Houston Instruments can also be used for output.

EasyFlow makes extensive use of the function keys (or the mouse menus and buttons) for chart editing. The drawing screen is divided into a narrow vertical window displaying the chart status, commands, and expanded view of the current chart element and a large, scrollable window for the entire chart. Zoom-in and -out commands are supported. Text can be placed either inside a shape or along a line. The chart editor supports block operations, such as move, copy, cut and paste, saving to disk, and loading chart fragments. EasyFlow also has an optional grid snap, like that of CAD programs, to align shapes. I also appreciated the ability to turn *off* the error bell. I wish more programs were this configurable.

The main-menu front end also includes such utilities as a file manager and an export utility. The export utility allows converting a saved EasyFlow chart to any of the following file formats: CGM (Computer Graphics Metafile), AutoCAD DXF, EPS (Encapsulated PostScript), GEM (used by Ventura), PLT (Hewlett-Packard HPGL), and TIFF (Tagged Image Format). The accompanying manual is very well written, with excellent illustrations and about the most detailed keyword index I've seen in a long time. If you need a tool to help you organize your thoughts before you start coding, EasyFlow could prove useful.

Books

Algorithms **by Robert Sedgewick**
Addison-Wesley Publishing Company
1 Jacob Way
Reading, MA 01867
800-447-2226
$40.95

This is the second edition of a fine book on algorithms. The author had Donald Knuth of Stanford as an advisor, through which his interest in algorithms was piqued. The first edition was a clear, well-written exposition on algorithms; in this edition, the writing is even clearer. The revised text is accompanied by hundreds of figures for visual clarity.

The focus of the book is on algorithms likely to be of practical use. The text is organized into eight major sections: Fundamentals, Sorting Algorithms, Searching Algorithms, String Processing, Geometric Algorithms, Graph Algorithms, and Advanced Topics.

The first section covers elementary data structures, trees, recursion, analysis of algorithms, and the implementation of algorithms. The section on data structures briefly covers arrays, linked lists, storage allocation, pushdown stacks, queues, and abstract data types. These chapters do an excellent job of introducing the topic in an organized and easy-to-read manner, and they set the tone and form a solid basis for the guts of the book—chapters on various algorithm types and their implementation.

The programming examples are provided in Pascal source code. Sorting and searching are well covered. Sorting methods described include selection sort, insertion sort, bubble sort, shellsort, quicksort, radix sorting, heapsort, mergesort, and external sorting. Some topics are covered at greater depth than others. For example, the chapter on quicksort, aside from implementing the basic algorithm, shows how it is possible to remove recursion. Various tree structures are described for searching—binary trees, indirect binary trees, top-down trees, 2-3-4 trees, red-black trees, and B-trees. I would have liked to see a more detailed discussion and implementation of B+ tree structures, since these are almost universally used for indexing in PC database applications. The coverage of string processing is solid, including brute-force methods, Knuth-Morris-Pratt, Boyer-Moore, Rabin-Karp, and multiple searches. The section on file compression is a bit weak, covering run-length encoding, variable-length encoding, and an implementation of Huffman code. The now common Limpel-Zev-Welch (LZW) algorithm is not mentioned. The sections on geometry, graphs, mathematical algorithms, and advanced topics are well done. Overall, I found this text a solid reference book for my algorithm shelf, and maybe you will, too.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

People are talking about us.

"Lahey F77L is definitely for the programmer's market, with features for the casual and professional user...*EDITOR'S CHOICE" PC Magazine*

"It is a fantastic product." *PC Australia*

"This is a very good implementation of FORTRAN, better than the one I use on my minicomputer system." *IEEE Software*

"It should be the last FORTRAN compiler you will need to buy..." *Your Computer*

"Lahey F77L was by far the easiest compiler to use..." *Programmer's Journal*

"Compilation speed is Lahey FORTRAN's most remarkable feature; it is unbelievably fast...Realistically, Lahey FORTRAN is the most efficient and productive FORTRAN development tool for the DOS environment..." *Byte*

F77L 4.0 EDITOR, LINKER, PROFILER GRAPHICS, MAKE, WEITEK

Contact us to discuss our products and your needs. (800) 548-4778
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123 Tlx: 9102401256

Number **44** on the Reader Service Card

NEW VERSION 4.5

DYNAMITE DEMOS

Proteus Prototype/Demo System

Why fizzle with an outdated demo program when you can explode with Proteus?

Proteus is the first prototype/demo system to combine complete simulation capability, a powerful Screen Painter, video effects, custom sound and much more in a program you don't have to be a genius to figure out.

And now Proteus supports all graphic modes for the CGA, EGA and VGA in the popular PCX file format.

"A joy to use."
. . . Computer Language

- ***Complete Simulation Capability***
- ***Powerful Screen Painter***
- ***Bit-Mapped Graphics***
- ***Video Effects***
- ***Custom Sound***

Don't Let Your Next Demo be a Dud!
Find out why software developers from Ashton-Tate to ZyLogic have switched to Proteus.

1-800-634-9986

$149. *No royalties. 30-day Money Back Guarantee. MC/VISA/COD/PO accepted.*

Helios

P.O. BOX 22869, SEATTLE, WA 98122

Number **258** on the Reader Service Card

DOS & OS/2
Free Demo Diskette Available

Q. How many programmers does it take to maintain a MAKE dependency file?

A. NONE! *If you use* Versi**MAKE**™

Versi**MAKE**™ is a full-featured MAKE utility that includes:

- **Dependency Generation**
  Derives your system's dependencies, through analysis of its C and MASM source files. Say goodbye to manual maintenance of MAKE dependency files!
- **Wild Card File Name Matching**
  Analyzes an entire collection of source files with a single statement.
- **Nested Include Files**
  Handles standard C and MASM "include" conventions, and the INCLUDE environment variable.
- **Powerful Macro Facilities**
  Built-in macros, user-defined macros, and environment variables.
- **Analytical Reports**
  Shows the entire include file hierarchy for each source file analyzed, and all of the parent source files for each Include file.

Q. How many programmers does it take to trace a symbol thru your system?

A. ONE! *If you use* Versi**CREF**™

Versi**CREF**™ is a unique utility that creates a Master Cross-Reference of your entire system.

- **Multi-Lingual**
  Handles C, assembler, or both.
- **Flexible**
  File names with line numbers, or file names alone. Global and local symbols, or globals alone.
- **Powerful**
  Easily handles systems containing 100 source files or more.

Versi**MAKE**™	$125
Versi**CREF**™	$75
Both	$150

(Outside U.S., add $5 for shipping and handling)

800-344-4096
(In NJ, 609-779-0784)
MC/VISA/AMEX accepted

SUMMIT INFORMATION SYSTEMS, INC.
496 N. Kings HWY Suite 130
Cherry Hill, NJ 08034

Number **158** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Software Tools
For Programmers & Non-Programmers

Opt-Tech Sort™

Opt-Tech Sort is a high performance Sort/Merge/Select utility. It can read, sort and write a file faster than most programs can read the data. Example: 1,000 records of 80 bytes can be read sorted and a new file written in less than 9 seconds (IBM XT). Opt-Tech Sort can be used as a stand-alone program or Called as a subroutine to over 30 programming languages.

Numerous features are included. A partial list of features includes: The ability to process files of any size. Many filetypes are supported including Sequential, Random, delimited, Btrieve, dBASEII & III and many others. Up to 30 key fields can be specified over 16 different types of data are supported. Powerful record selection capability allows you to specify which records are to be included on your output. Record reformatting allows you to change the structure of your output record and to output special fields such as record numbers for use as indexes. **Newest Features: Faster than ever! LIM EMS support, User exits to pass or receive records. Less disk space required and much more!** MS-DOS $149.

On-Line Help™

On-Line Help allows you to easily add Help Windows to all your programs. On-Line Help is actually two help packages in one. You get BOTH Resident (pop-up) and Callable help systems.

The resident version allows you to add help to any system. Your Help system is activated when the "Hot Keys" that you specify are pressed. You can then chain between help windows in any manner you desire. Upon clearing the help windows you are right back into what ever program you left.

The callable version allows you to easily display help windows from your programs. A simple Call to the help system makes the window appear. On-Line Help is callable from over 20 different languages.

You have full control over the help window content, size, color and location. MS-DOS $149. Demo $10.

Scroll & Recall™

Scroll & Recall is a resident screen and keyboard enhancement. It allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations). Also allows you to easily recall and edit previously entered DOS commands without retyping.

Scroll & Recall is very easy to use. Its a resident utility that's always there when you need it. MS-DOS $69.

Visa, M/C, AMEX, Check, Money Order, COD or Purchase Orders accepted

To order or to receive additional information just call and receive immediate highly qualified attention!

Opt-Tech Data Processing
P.O. Box 678 / Zephyr Cove, NV 89448
(702) 588-3737

Number **16** on the Reader Service Card

MODULA-2 PROGRAMMERS

Being squeezed by the DOS 640K limit? Need more heap space? PowerSTOR uses any combination of expanded memory, extended memory and hard disk space available on the target machine. It self-configures at run time under your control. Use multiple heaps, as big as you want. PowerSTOR is fast and easy to use. Available for the Taylor, Logitech and JPI Modula-2 compilers.

Price $57.50 (includes shipping), with source code $87.50.

Acme Software
P.O. Box 40734
Portland, OR 97240
503-235-2816 **RS# 305**

BTRIEVE ALTERNATIVE

Application programmers: Get a secure, multilanguage, multiuser, full, flexible indexed file manager: AXOS, only $395. "AXOS provides 57 functions, including audit trail and key-only file management functions, and 64 index paths vs. Btrieve's 31 operations and 24 access paths." -*PC TECH JOURNAL* (Oct 1988)

McSoft, Inc.
P.O. Box 12044
Atlanta, GA 30355
404-984-0608 **RS# 230**

WHITNEY EDITOR $39

Small and fast. Uses all available memory. Split screen editing. Configurable keyboard. Regular expression search. One key compile. Features for writing documentation. Condensed/Outline display. Runs on IBM PC's, AT's, and PS/2's.

USA shipping and handling $3. Outside USA $15. CA residents add sales tax.

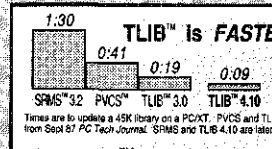
Whitney Software, Inc.
P.O. Box 4999
Walnut Creek, CA 94596
415-933-9019 **RS# 298**

CSOURCE APPLICATION GENERATOR

C code WYSIWYG screen, report, & program generator. Library code may be enhanced for custom generation. Supports multi-user development & DOS 3.1/Novell networks. Produces small sized DATA ENTRY, REPORT, MENU, INDEX REBUILD & FILE RESTRUCTURE programs with multiple screens & files. Very fast development & execution speeds. Requires 256K RAM, DOS 2.1 or later, Faircom Ctree or Novell Btrieve. For Turbo or Microsoft C 5.0. No royalties. ALL generated applications source is supplied.

$395.00 executable, $595.00 full source code. Multi-programmer licenses.

MBAC
13320 Castana Ave.
Downey, CA 90242
213-531-7259 **RS# 291**



TLIB™ is *FASTEST!*

Times are to update a 45K library on a PC/XT. PVCS and TLIB 3.0 are from Sept 87 *PC Tech Journal*. SRMS and TLIB 4.10 are later versions.

TLIB™ is *BEST!*

"Do not be fooled by the fact that this is the least expensive of the five packages reviewed here - TLIB has features and power to spare" John Rex, **Computer Language**

"TLIB is a great system" J. Vallino, **PC Tech J**

- **Full-Featured Version Control for Software Professionals.** *Check-in/out locking. Branching* Keywords. Wildcard and list-of-file support. Can merge parallel changes and undo intermediate revisions. Network and WORM support. **Mainframe compatible** deltas for Pansophic, ADR, IBM, Sperry. Integrated Landon Dyer PD **MAKE**.

MS-DOS 2.x, 3.x, 4.x ***Just $99.95*** + $5 s/h Visa/MC 5 station LAN license $299.95 + $5 s/h, call for other sizes

BURTON SYSTEMS SOFTWARE
PO Box 4156, Cary, NC 27519 **(919) 856-0475**

Number **288** on the Reader Service Card

Network API Library
For *C* or *Pascal*, supports both *Microsoft* and *Turbo* compilers, Compatible with *IBM, 3Com, Banyan*, and *Novell* Networks. Includes DOS network functions, **NetBIOS, IPX/SPX**, Client/Servers, **Remote Procedure Calls**, Remote screens/windows. $99.95

NetWorkStation Utilities
Network Consulting, Distributed Processing, Classroom Screen Broadcasts, Work Groups, Device Sharing. NetBIOS/Netware $99.95

Information Software Associates
Box 2582, Iowa City, IA 52244
Call for more info **(319) 354-6804**

Number **294** on the Reader Service Card

***PJ* 7.5 ISSUE SPECIAL**

$5.00 OFF
***PJ* Code DiskScription**

GET MORE CODE FOR LESS COST!

Get all the program listings (plus additional .EXE, .OBJ, and benchmark source files) published in the next 6 issues of *PJ*. See ads on pages 8 and 88 for more details.

For fastest service call toll free
800-234-0386
(All orders must be prepaid. Offer expires 11-31-89.)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

YOUR CUSTOMERS DESERVE THE BEST - CUT YOUR SUPPORT COSTS

Install your software as if you were there yourself, adapting to hardware, software, environment and installer wishes. Just "fill in the blanks" to create multiple adaptable options in minutes, intelligently controlling AUTOEXEC, CONFIG and much more. New installations, updates, MODS. 250-page User's Guide, 45 pages on automatic handling of special and abnormal situations.

INSTALLATION TOOLKIT executables $49.00. With C or TP 5.0 tools, $99.00. No royalties. VISA/MC.

Helpful Programs, Inc.
P.O. Box 16078
Huntsville, AL 35802-1627
800-448-4154 **RS# 280**

dBASE FILE ACCESS FROM C

Code Base 4 is a library of C routines which gives complete dBASE or Clipper functionality and file compatibility. The routines correspond directly to dBASE commands. Build a multi-user, 85K dBASE compatible application using pulldown menus, popup windows, and data entry from pick lists. Use DOS, Unix, OS/2 or Microsoft Windows. A dBASE to C translation utility is available.

$295 with source! FREE DEMO

Sequiter Software Inc.
P.O. Box 5659, Station L
Edmonton, Alberta
Canada T6C 4G1
403-439-8171
Fax: 403-433-7460 **RS# 269**

DOCUMENTATION COMPILER

EASILY CREATE AND SHARE SOURCE CODE DOCUMENTATION. DocuMotion builds cross-indexed project documentation libraries from your source code comments. Language-Independent. Resident/nonresident view utility gives instant shared access to all project documentation, manuals, etc. powerful print and copy capabilities. Folding documents, electronic bookmarks, search, LAN-compatible, more. CALL FOR FREE INFORMATION KIT.

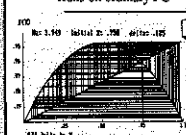
Personal version $149, 3 station network $359, 6 station network $597.

NWP-Intelligent Solutions Inc.
P.O. Box 20478
Bloomington, MN 55420-0478
612-884-5860 **RS# 283**

CHAOS Manner

CHAOS Manner is an exciting introduction to the new science of chaos. It introduces important concepts without using non-linear differential equations.

- Powerful, interactive graphics
- Spreadsheet-type user interface
- 256x magnification facility
- Course and fine display modes
- Rigorous treatment without the math
- Discusses chaos in neural nets
- Complete source code included
- Fast PL/D compiler, 90+ page manual
- Print graphics with DOS utilities
- Tutorial manual
- Runs on ordinary PC **$97.95**



DAIR Computer Systems
3440 Kenneth Drive
Palo Alto, CA 94303
Tel: (415) 494-7081
FAX: (415) 494-8831
Munich: (089) 780-9608
Tokyo: (03) 222-0541

Add shipping and handling of $5.00 in the U.S.A. ($12.50 overseas) and sales tax in California. Needs PC, AT, or PS/2, or compatible; 256K; DOS 2.0+; CGA, EGA, VGA, or compatible adapter.

Number **5** on the Reader Service Card

Mktg/Sales Team For Hire

Watch Your Sales Soar!

We help software publishers and hardware manufacturers get the most out of their limited marketing dollars. Let *Techsel* help you market/sell your product - we can *guarantee* your satisfaction. Call Ed Harms or Stewart Gittelman at 516-277-7000.

***Techsel* Corporation**

Number **264** on the Reader Service Card

PJ CODE IS AVAILABLE

ON DISK—CALL
800-234-0386
$10.00 each, VISA/MC

See ads on pages 8 and 86 to order, or see pages 61 and 73 for information on BBS downloads.

PJ Code DiskScription
ISSUE SPECIAL 7.5
$5.00 off when you subcribe today!
(see ad on page 88)

DO IT YOURSELF!

XenoCopy-PC

PC-DOS program lets your PC
Read/Write/Format
over 350 formats **$79.95**

XenoFont

high quality text screen printouts
$49.95

+$5.00 S/H+ Tax if CA.

XenoSoft • (415) 644-9366
2210 Sixth Street,
Berkeley, CA 94710

Number **123** on the Reader Service Card

"QUICK & CLEAN" COBOL

The COBOL Report Activator (COBRA):

- Generates efficient, structured COBOL programs
- Is easy to learn and use
- Accepts your COBOL code and COPY members
- Produces audit trail reports
- Up to 5 output files per program
- Provides report options & site defined defaults
- Is available for Mainframe & PC

COBRA
Systems Software
David Olivadoti, P.O. Box 750817
Petaluma, CA 94975
(707) 763-5123

Number **281** on the Reader Service Card

JAKE™: NATURAL LANGUAGE PROCESSING USING C

Call JAKE Routines To Convert Sentences Into C Data Structures. Easy To Build Application Dictionary.

OUR CUSTOMERS USE JAKE FOR

COMMANDS
Scripts
Process Control Systems
Expert Systems
Command Languages

QUERIES
Hypertext Systems
Help Systems
Ad Hoc Database Queries

JAKE $495. Interactive Demo $10. CALL (408) 438-6922 VISA, MC

EKS English Knowledge Systems, Inc.
5525 Scotts Valley Dr. Suite 22
Scotts Valley, CA 95066

Usable with Microsoft C, Turbo C, OS/2, and Windows. Context-sensitive semantic processing. Easy interface to any application. Mem: 64K.

Number **243** on the Reader Service Card

FREE CATALOG!

LATEST AWARD BIOS
PC/XT, 286, 386

COMPILED BASIC TOOLS
Libraries, Utilities

COMPUTER BOOKS
Graphics, OS/2, Assembly

HARDWARE UPGRADES

(800) 423-3400
KOMPUTERWERK INC.
851 Parkview Blvd.
Pittsburgh, PA 15215

Number **155** on the Reader Service Card

C SOURCE CODE

FOR ACCOUNTING BUSINESS AND MANUFACTURING MANAGEMENT

XLN offers K&R source code modules for general ledger, sales and accounts receivable, purchasing and accounts payable, inventory, bill of materials, MRP, CRP, PERT, maintenance management, personnel and payroll, and more.

XLN SOFTWARE
3680 Tuggle Road
Buford, GA 30518
800-543-5770
FAX: 404-859-0051
GA, Int'l or support: 404-945-8745

Number **237** on the Reader Service Card

PCHRT V3.00

HIGH RESOLUTION TIMER TOOLBOX

The definitive answer to execution profiling and embedded timer problems. 30 functions manage 100 software timers with microsecond accuracy. Self calibrating. Extensive and flexible timer reporting. Profiles selected ROM BIOS and MSDOS interrupts.

Supports TP 4 & 5, TC, MSC 5.x and QC 2.0. Libraries, manual, examples, and full source included.

$24.95 ppd USA, elsewhere add $4.00, Mich add 4%. VISA/MC

RYLE DESIGN
P.O. Box 22
Mt. Pleasant, MI 48804
(517) 773-0587

Number **210** on the Reader Service Card

PC-lint

C Programmers: Locate bugs, glitches, quirks and inconsistancies in your C code. PC-lint supports multiple modules, K&R, ANSI. Specify OS/2 or MS-DOS. Only $139.

Gimpel Software
(215)584-4261
3207 Hogarth Ln
Collegeville PA 19426
30 Day Money-back Guarantee

Contact this advertiser directly.

MIDNIGHT ENGINEERING™

A new magazine for entrepreneurial hardware and software engineers that will encourage and challenge you to personally develop and market your own products.

- **PRACTICAL ARTICLES**
- **INSIGHTFUL INTERVIEWS**
- **DETAILED PRODUCT REVIEWS**

This is your chance to influence a brand-new magazine. PJ readers are invited to a **FREE** copy of the premiere issue of *Midnight Engineering* by calling or writing us with ideas.

DATArx
111 E. Drake Rd., Suite 7041
Fort Collins, CO 80525
303-223-2120

Number **272** on the Reader Service Card

PRODUCED 2024 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
282	Acucobol Inc.	25
305	Acme Software	94
30	The Austin Code Works	53
207	Automation Education	86
51	Az-Tech Software, Inc.	55
303	BBI Company	35
244	Block Island Technologies	86
3	Borland International	C4
288	Burton Systems Software	94
286	Business Computer Services	86
299	Clarion Software	1
*	CNS, Inc.	47
281	COBRA Systems Software	95
301	Complete Logic Systems	29
277	Covox Inc.	48
5	Dair Computer Systems	95
272	DATArx	95
229	Disk Software, Inc.	35
241	Emerald City Software	7
243	English Knowledge Systems	95
267	Falk Data Systems	73
307	F G Soft Company	47
261	Genus Microprogramming	28
*	Gimpel Software	95
*	Graphic Software Systems	35
248	Hammerly Computer Services, Inc.	90
293	Hammerly Computer Services, Inc.	91
122	Hauppauge Computer Works, Inc.	89
42	Harvard Softworks	41
154	HavenTree Software Limited	15
258	Helios Software	93
280	Helpful Programs	95
294	Information Software Associates	94
226	Ithaca Street Software, Inc.	61
295	Knowledge Dynamics Corporation	65
155	Komputerwerk	95
63	Laboratory Microsystems Incorporated	81
44	Lahey Computer Systems Inc.	93
246	Logical Alternatives Incorporated	55
271	Lugaru Software Ltd.	71
291	MBAC	94
230	McSoft	94
143	Metagraphics Software Corporation	30
45	MetaWare Incorporated	59
232	Microsoft Corporation–Human Resources	9
*	Microsoft Press	27
104	Microway	43
121	MISOSYS, Inc.	49
266	Mix Software	84
94	MMC AD Systems	14
109	Nanosoft	73
302	NTERGAID	10
160	Nu-Mega Technologies	3
283	NWP Intelligent Solutions	95
16	Opt-Tech Data Processing	94
270	OTG Systems Inc.	48
*	Paradigm Systems	81
208	PC Tech	31
6	The Periscope Company	19
*	***PJ Code DiskScription***	8
279	Pocketsoft	13
*	***Programmer's Journal*** Mail Order Store	88
114	Programmer's Shop, The	11
228	ProTech Marketing, Inc.	60
97	Qualstar Corporation	55
252	Quantasm Software	65
147	Quarterdeck Office Systems	C2
296	Que Corporation	37
222	Rainbow Technologies	33
20	Rational Systems, Inc.	85
210	Ryle Design	95
269	Sequiter Software Inc.	95
165	SLR Systems	10
33	Soft Advances	50
167	Software Development '90	C3
275	Software Development Technologies	42
98	The Software Engineering Store	67
198	Software Science, Inc.	77
221	Software Security	5
158	Summit Information Systems, Inc.	93
297	Talking Technology, Inc.	6
289	TechnoJock Software Inc.	76
264	Techsel Corporation	95
166	Tool Makers, The	22
276	Tri-Technology Systems Inc.	42
25	TurboPower Software	79
290	Ultra-X	66
125	V Communications, Inc.	69
287	Vestronix, Inc.	61
304	Visitech Software	31
100	Western Wares	58
73	The Whitewater Group	38
298	Whitney Software, Inc.	94
123	Xenosoft	95
237	XLN Software	95

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers with an * prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors.

COMING ISSUES—

November/December 1989 Volume 7.6

User-Interface Design:

A Simple Terminal Program for Windows *by William S. Hall*

Using the DESQview API *by Todd Hoff*

ASM:

Efficient 80x86 Memory Models and Static Variables *by John Otken*

On Graphics:

Split Screens on the VGA *by Michael Abrash*

January/February 1990 Volume 8.1—Networking

Programmer's Journal
November/December 1989 Vol. 7.6
Advertising Space Reservation Deadline:
September 8, 1989
Materials Due:
September 22, 1989
Please call *Programmer's Journal* to reserve your space.
(503) 747-0800

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"The only serious technical conference of note for the PC community."

Garry Ray/PC Week

If you can attend only one conference this year, don't miss Computer Language's SOFTWARE DEVELOPMENT '90 — February 6-9, 1990 at the Oakland Convention Center.

A different kind of conference — by and for programmers

At SOFTWARE DEVELOPMENT '90 you'll have three days to focus on practical techniques that will increase your programming power. You'll learn from the most important visionaries and practitioners in software development today. Past speakers include: Larry Constantine, Philippe Kahn, P.J. Plauger, Bill Gates, Bjarne Stroustrup, Ed Yourdon, Steve Mellor, Rich Finkelstein, Robert Ward, Jack Purdum, and dozens more.

Hone your professional skills, investigate new programming tools

Imagine the chance to pick and choose from over 150 lectures and workshops on design methodologies, programming tools and issues, object-oriented programming, C and other programming languages, artificial intelligence, graphics interface design and much more, all tailor-made for you — the experienced software developer.

Share tips and techniques with the "best and the brightest" at organized roundtable discussions, informal evening receptions and vendor hospitality events.

The very latest in development tools and services will be on display at the concurrent Software Development Products Exhibition. Over 100 exhibitors make it the largest trade show exclusively for Software Developers.

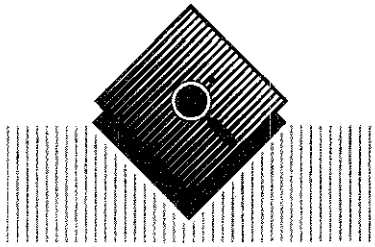
Just minutes from downtown San Francisco

The Oakland Convention Center gives you complete access to the exciting San Francisco Bay Area. From the front door of the center, you're 7 minutes from downtown San Francisco by BART train.

"Probably the most valuable thing I've done in ten years in the software business."
Chuck Atkinson/CAP, Fort Worth, Texas

Register early and save up to 30%

Program details are now being finalized. Return the coupon today for a free information kit. Or call Anastasia Mills at (415) 995-2471. Learn how you can save up to 30% by registering early.



SOFTWARE DEVELOPMENT '90

Oakland Convention Center
February 6-9, 1990

Produced and managed by the publishers of Computer Language, AI Expert, UNIX Review, Database Programming and Design, and Embedded Systems Programming.

Software Development '90
Miller Freeman Publications
500 Howard St., San Francisco, CA 94105
Phone: (415) 995-2471 • FAX: (415) 995-2486

YES!

Please send me a free information kit and registration materials for SOFTWARE DEVELOPMENT '90 to be held February 6-9, 1990 at the Oakland Convention Center.

☐ Please send me information on exhibiting.

PJ 9/10

Name _____

Title _____

Company _____

Street _____

City, State, Zip _____

Phone _____ FAX _____

Software Development '90, Miller Freeman Publications, San Francisco, CA 94105

Number **167** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BORLAND INTRODUCES TURBO PASCAL 5.5 WITH OBJECTS

Be Objective.

Turbo Pascal,® the world-standard Pascal compiler, adds Object-Oriented Programming with our new version 5.5. We combined the simplicity of Apple's Object Pascal language with the power and efficiency of C++ to create Turbo Pascal 5.5, the object-oriented programming language for the rest of us.

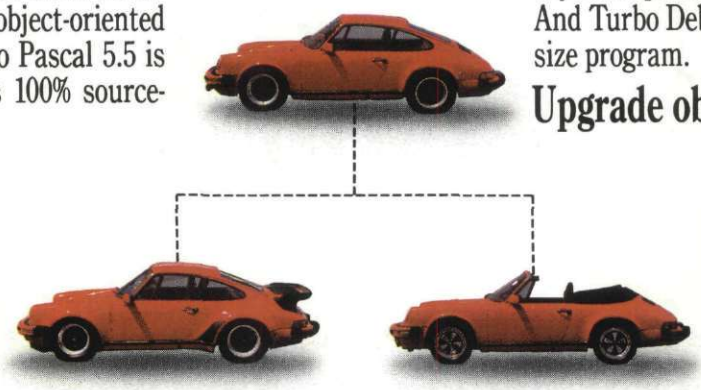
It's easy to extend yourself

If you're already programming with Turbo Pascal, it's easy to extend yourself from structured programming to object-oriented programming. And, Turbo Pascal 5.5 is the *only* compiler that is 100% source-code compatible with your existing Turbo Pascal 4.0 and 5.0 programs.

A fast object lesson

Object-oriented application programs more closely model the way you think. Objects contain both data and code. As in a spreadsheet cell, the value and the formula are together. Objects can *inherit* properties from other objects. For example, a Porsche Carrera inherits most attributes from the base model 911, but it also sports a whale tail.

Turbo Pascal 5.5's object-oriented extensions give you code that's easier to change, extend, and support.



Inheritance provides powerful modeling capabilities by allowing objects to inherit attributes from other objects.

Turbo Pascal 5.5 Professional with Turbo Debugger® and Turbo Assembler®

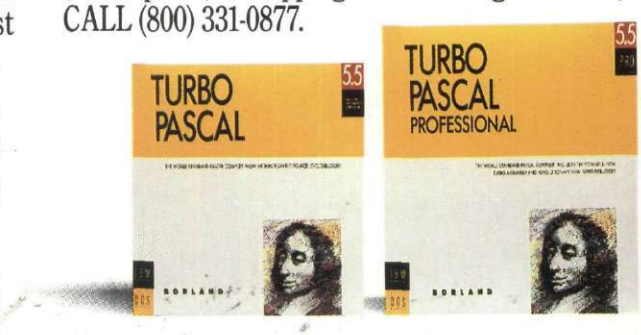
The award-winning Turbo Debugger now includes an object inspector and hierarchy browser. And Turbo Debugger can debug any size program.

Upgrade objectively

Pascal owners: Upgrading from Turbo Pascal 5.0 to 5.5 is only $34.95 plus $5 shipping and handling ($75 plus shipping and handling for owners of Turbo Pascal 4.0 or earlier). And upgrading from Turbo Pascal 5.0 and earlier to Turbo Pascal 5.5 Professional is only $99.95 plus $10 shipping and handling. To order, CALL (800) 331-0877.

Turbo Pascal 5.5 Features

- Inheritance
- Static & dynamic objects
- Constructors & Destructors
- Object constants
- Compiles @ > 34,000 lines/minute
- New integrated environment tutorial
- Hypertext Help with copy and paste
- Enhanced smart linker & overlay manager
- Support for 8087/80287/80387
- Integrated source-level debugging



BORLAND

CODE: MA27

Mail upgrade orders to: Borland, P.O. Box 660001, Scotts Valley, CA 95066-0001. For orders outside the U.S., call (408) 438-5300.

Turbo Pascal, Turbo Debugger, and *Turbo Assembler* are registered trademarks of Borland International. Copyright ©1989, Borland International, Inc. All right reserved. BI 1324

Number **3** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED